THE Open GROUP

Agile Architecture in the Digital Age

A White Paper by:

Hervé Barbazange, Société Générale
Peter Beijer, DXC Technology
Jean-Marc Bunouf, Société Générale
Carl Kinson, DXC Technology
Frédéric Lé, DXC Technology
Jean-Pierre Le, Société Générale
Antoine Lonjon, MEGA International
Jérôme Régnier, Societe Générale

With special acknowledgement for advice and support from:

Xavier Cabot, Cesames
Daniel Krob, Ecole Polytechnique and Cesames

July 2018

Agile Architecture in the Digital Age
Document No.: W186

Published by The Open Group, July 2018.
Any comments relating to the material contained in this document may be submitted to:
The Open Group, Apex Plaza, Forbury Road, Reading, Berkshire, RG1 1AX, United Kingdom
or by email to:
ogpubs@opengroup.org

Table of Contents

Boundaryless Information Flow ™
achieved through global interoperability
in a secure, reliable, and timely manner

Executive Summary

The effectiveness of agile processes is too often jeopardized because the architecture and organizational pre-requisites of agility are neglected.

This White Paper proposes a new architecture framework, the Agile Architecture Framework (AAF), that meets the needs of the digital enterprise. It develops a vision that combines in a unique manner:

- Methods for decomposing the system, and the organization that designs it, into loosely-coupled services and autonomous teams
- Alignment mechanisms rooted in business strategy that promote a shared culture that becomes the glue that keeps empowered organizations from falling apart
- Architecture patterns that leverage the latest software innovations in distributed computing, autonomous systems, data streaming, and artificial intelligence
- Validated learnings from very large enterprises that have started their agile-at-scale journey a few years ago

This proposed architecture approach will enable organizations at all scales to better realize the Boundaryless Information Flow™ vision achieved through global interoperability in a secure, reliable, and timely manner.

Introduction

We observe that current architecture practices and skills come under scrutiny because they are typically anti-patterns of a lean and agile culture. They are too often perceived to stand in the way of iterative development, Minimum Viable Products (MVPs), and collaboration.

The Architect today will pull on years of tradition, but will have to operate in a different way, creating new artifacts, learning new skills, and working as members of cross-functional teams. Architecture needs to create usable assets that resonate with engineering and operations teams.

This White Paper aims to address these issues and start to lay out the new architecture framework that the digital enterprise needs.

A recent McKinsey survey[1] shows that organizational agility is on the rise: "*the need for companies to demonstrate agility is top of mind*". An increasing number of large firms are deploying agile-at-scale frameworks, such as the Scaled Agile Framework® (SAFe®),[2] Large-Scale Scrum (LeSS™),[3] or the Spotify® Model.[4]

A paper published in IEEE Software[5] claims that process improvement alone cannot fix the root causes of poor agility: "Agile practitioners have focused intensely on improving software development processes and not so much on technical health … We've worked with several large organizations in which the application of lean principles produced underwhelming results … This is because velocity measurement, planning poker, attacking defect backlogs, Kanban cards, pair programming, or sprint-based planning do little to attack the root cause of problems that are inherently structural."

Experiences from the field confirm the aforementioned: root causes are not limited to technical health, they also originate in the organization and culture of the enterprise. The verbatims below illustrates this:

- *"Legacy methods tend to slow down the initial sprint … Subsequent sprints often change architecture models defined in previous sprints."*
- *"If we're going to have to do a heavy architecture which plans for a year or two or five years into the future on every one of those experiments, we're in serious trouble. We cannot be agile."*
- *"... We want to be able to put in the smallest, simplest, minimum viable experiment, prove an assumption, beef it up if we want to or follow it wherever it goes, pivot and follow it wherever it goes. That means our entire architecture is going to be emergent based on where we want to go."*

[1] See www.mckinsey.com/business-functions/organization/our-insights/how-to-create-an-agile-organization.

[2] See www.scaledagileframework.com.

[3] See https://less.works/less/framework/introduction.html.

[4] Scaling Agile @ Spotify with Tribes, Squads, Chapters & Guilds, Henrik Kniberg, Anders Ivarsson, October 2012; refer to: https://blog.crisp.se/wp-content/uploads/2012/11/SpotifyScaling.pdf.

[5] Modular Architectures Make You Agile in the Long Run, Dan Sturtevant, IEEE Software, Vol. 35, Issue 1, January/February 2018; refer to: https://ieeexplore.ieee.org/document/8239949/.

- *"... Ideally that pool of seniors in your team act as a kind of proxy architecture committee, and we don't have to go to someone who's supposedly got the title sitting in an ivory tower, and has never actually built that thing in the last ten years because they've been thinking high-level."*
- *"At an application level I think that those architects are a waste of time. I really don't think they know what they're talking about nowadays."*
- *"Though the architecture discipline is needed, the architect's role as a squad member is ill-defined ... Squad architects must attend all agile ceremonies, otherwise they are at risk of becoming marginalized."*

It is our firm belief that architecture (the thing) and architecting (the verb) cannot be treated separately. Put differently, in an agile context technical health and the process dimension go hand-in-hand. Enterprises too often have neglected the architecture, organizational, and cultural pre-requisites of agility as the primary focus of agile transformations has been the process dimension.

Generally, we see that:

- Enterprise Architects should (re)focus their attention on modularizing monolithic systems, because it is the number one pre-condition for agility
- Existing architecture practices and roles need to evolve to remain relevant in an organization that adopts agile ways of working
- The body of knowledge of architects needs to be completed to meet the needs of the digital enterprise
- Classical architecture governance models lose relevance when shifting from large programs toward multiple autonomous teams

This White Paper formulates a vision that has the ambition of solving these problems. It is based on the diagnostic below:

- When teams are not autonomous enough, it slows down continuous delivery which limits agility
- To avoid chaos, team autonomy must be balanced by alignment mechanisms that cannot rely on a command-and-control culture that otherwise would get in the way of autonomy
- New software architecture patterns deeply influence the evolution of Enterprise Architecture
- The digital enterprise needs a new architecture body of knowledge, new processes, and governance practices; architecture roles need to be redefined

Autonomy and Loose-Coupling

The 2017 State of DevOps Report[6] investigated how architecture correlates with continuous delivery and IT performance. They measured coupling between services and components by capturing whether:

- Respondents could do testing without requiring an integrated environment
- Applications and services could be deployed or released independently of other applications and services on which they depend

The 2015 DevOps report discovered that high-performing teams were more likely to have loosely-coupled architectures than medium and low-performing teams. The 2017 DevOps report confirmed this and verified two new hypotheses:

- Teams that can decide which tools they use do better at continuous delivery – this is in contrast to teams that can use only those tools that are mandated by a central group
- In teams with strong IT and organizational performance, the architecture of the system is designed so delivery teams can test, deploy, and change their systems without depending on other teams for additional work, resources, or approvals, and with less back-and-forth communication

The key takeaway is that when systems and teams are too coupled, the process is stuck in what Forrester labels as "Water-Scrum-fall".[7] Agile development cycles can be rapid, but deployment is slowed because of all the coordination, testing, and integration work that is still required before a service can be deployed into production. The net is that Water-Scrum-fall has lead times that are not much shorter than waterfall ones, defeating the purpose of agility which is to deliver value faster and more often.

Academic research by MacCormak et al.[8] demonstrates that a relationship exists between the structure of an organization and the design of the products that the organization produces. A natural experiment shows that loosely-coupled organizations develop more modular designs than tightly-coupled organizations.

The authors compared six pairs of similar products. Each pair is composed of a product developed by a loosely-coupled organization and the other by a tightly-coupled organization. A product developed by a loosely-coupled organization was significantly more modular than a product developed by a tightly-coupled organization. Modularity was measured by capturing the level of coupling between a product's components.

The magnitude of the differences was substantial – up to a factor of eight, in terms of the potential for a design change in one component to propagate to others.

[6] See www.ipexpoeurope.com/content/download/10069/143970/file/2017-state-of-devops-report.pdf.

[7] Analyst Watch: Water-Scrum-fall is the reality of Agile, Dave West, December 2011; refer to: https://sdtimes.com/agile/analyst-watch-water-scrum-fall-is-the-reality-of-agile/.

[8] Exploring the Duality between Product and Organizational Architectures: A Test of the "Mirroring" Hypothesis, Alan MacCormack, John Rusnak, Carliss Baldwin, Harvard Business School Working Paper; refer to: www.hbs.edu/faculty/Publication%20Files/08-039_1861e507-1dc1-4602-85b8-90d71559d85b.pdf.

The key takeaway is that to architect a loosely-coupled system, it is important to pay attention to the organization that will produce it. Because the two are congruent, the reverse Conway law[9] suggests that the design of the architecture should influence the design of the organization.

That relationship being established, we will explore how to decompose a software system and the organization that will produce it.

Traditionally, architecture has focused on layering software systems based on technology concerns such as data access, business logic, application logic, or presentation logic. The main benefits are:

- Standardization which could limit technology risks and leverage economies of skills
- A form of modularity because changes in one layer do not impact layers below

Layering has created a generation of architects motivated by learning technologies to increase their market value, and less interested in learning the domain. In his seminal book,[10] Eric Evans claims that the domain is the main source of software complexity. He has developed a method, Domain-Driven Design (DDD), to address it.

Domain-Driven Design

In a nutshell, Domain-Driven Design (DDD) decomposes the domain into sub-domains and contexts. If done well, the resulting domain architecture defines a set of loosely-coupled services. DDD focuses the attention on the vertical decomposition of the system. Figure 1 illustrates the shift.

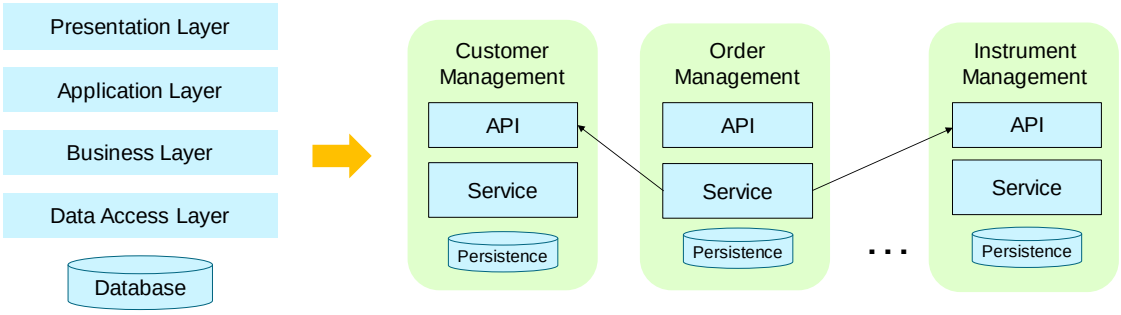

Figure 1: From Layers to Services

The system is decomposed into services that encapsulate a set of homogenous capabilities. The modularity rule applies, cohesion is high within a service, inter-service coupling is low, and implementation details are hidden behinds APIs.

Each service owns its persistence mechanisms and exposes its functions and features through well-defined interfaces. Inter-service communication occurs through synchronous or asynchronous interactions. When

[9] See www.agilealliance.org/resources/sessions/the-reverse-conway-organizational-hacking-for-techies.
[10] Domain-Driven Design: Tackling Complexity in the Heart of Software, Eric Evans, Addison Wesley, August 2003.

communication is asynchronous, messages or events link services through protocols such as the publish-and-subscribe one. No communication is allowed through database sharing, shared libraries, or other mechanisms.

The choice of tools and development stack is not constrained, which has two upsides:

- Innovation is not slowed down by technology standards that are likely to become obsolete over time
- Teams that can decide which tools they use do better at continuous delivery[11]

Services can be tested and deployed in isolation and are easily containerized, which helps speed continuous deployment.

Decomposing a domain requires deep domain knowledge to avoid designing services that expose APIs prone to abstraction leaks. The level of granularity of services can vary. When services are responsible for a single capability, they are referred to as microservices.

Choosing the right level of responsibility for each service – its scope - is one of the most difficult challenges."[12]

DDD is now a well-adopted approach to help decompose a system into modular parts.

"Many microservice adopters have turned to Eric Evans' "Domain-Driven Design" (DDD) approach for a well-established set of processes and practices that facilitate effective, business-context–friendly modularization of large complex systems."[13]

However, even a good method cannot replace domain expertise, therefore Martin Fowler[14] advises to start with a monolithic implementation and refactor it into microservices when the domain is better understood.

"Eventually the team merged the services back into one monolithic system, giving them time to better understand where the boundaries should exist. A year later, the team was then able to split the monolithic system apart into microservices, whose boundaries proved to be much more stable."[15]

Toward Modular and Empowered Organizations

Let us now look at how enterprises alter their operating models and organizational structures to become more agile and how it is congruent with the evolution of software systems toward modularity.

The traditional way of steering change relies on programs and projects staffed from shared pools of resources. You could set up a marketing project to define the product, an IT project to develop supporting applications, and a change management project to steer deployment within the organization. Engineering teams would send software "over the wall" to IT operations.

[11] Source: 2017 State of DevOps Report

[12] Microservices in Action, Morgan Bruce, Paulo A. Pereira, Manning Publications, 2018.

[13] Microservice Architecture: Aligning Principles, Practices, and Culture, Irakli Nadareishvili, Ronnie Mitra, Matt McLarty, Mike Amundsen, O'Reilly Media, 2016.

[14] See https://martinfowler.com/articles/microservices.html.

[15] Building Microservices, Sam Newman, O'Reilly Media, 2015.

In the new operating model, the focus is shifting toward stable teams with dedicated resources that are responsible for designing, building, and running products or services. The process is managed by strong product owners, often from the business, who work closely with IT at all stages of the product lifecycle.

All roles are integrated within self-organizing feature teams or squads sometime regrouped into tribes.[16] The project manager role is shifting toward an agile coach role and line managers focus on capability building.

A Practical Example

We will illustrate this with an ecommerce enterprise whose business model is based on sales that last for a few days and are announced only 24 hours in advance. Brands only appear twice a year … By frustrating demand and putting on time constraints, it aims to create desire and impulsive buying.

Though service quality is sub-standard, the business model works well and has generated consistent high growth over the last ten years. In the meantime, Internet giants have influenced customer expectations. For example, clients want to know estimated shipping delays, they need payment flexibility, and wish they could regroup several items into one shipment if purchased the same day.

The enterprise is continuously experimenting with new ways of selling products; for example, 3D or 360° UIs on the media side. The fast-growing scale of operations makes each evolution slower and more difficult. Nothing significant can now be accomplished without effective technology support.

A new CTO has been appointed by the CEO. He has the mandate to put back technology at the center of the enterprise. Yesterday it was a fashion start-up, today it is becoming a tech company. Yesterday the IT department was organized in a pyramidal manner like an IT services company, but now the organization is becoming distributed.

The enterprise and the software system are being decomposed into 50+ products managed by autonomous teams; for example, a payments team and a logistic team whose missions are respectively to create and run the best payments or logistics products.

The current software system which is too inflexible and monolithic is being re-built in an incremental manner using the microservices architecture style.

The leadership team wants to remove any obstacle that could slow down these autonomous teams. The purpose of the new organization is to regain the agility of a startup while operating at the scale of a multi-billion business. A sense of urgency drives the change agenda because the CEO feels that: *"a business can die from non-controlled high growth"*.

Even though teams are highly autonomous, alignment is needed. To be more specific, the enterprise:

- Must ensure that the journey of a customer remains frictionless and provides an experience at par with the one Internet giants provide
- Needs to consolidate data from various sources to develop highly effective predictive models – the

[16] See http://schd.ws/hosted_files/agilecamppacificnorthwest2017/20/AgileCamp2017%20-%20Jeff%20Nicholls.pdf.

business model requires building inventory prior to sales, therefore marketing and sales data are critical to predict the demand that logistics must fulfill

How can such an enterprise walk the fine line that separates chaos from "empowered" alignment?

Balancing Autonomy with Alignment

"*Ensure that when the bottom spoke the top listened – was one of the challenges we would eventually have to overcome … Order can emerge from the bottom up, as opposed to being directed, with a plan, from the top down."*[17]

The separation of decision-making from work characterizes command-and-control thinking. It keeps managers out of touch of their operations. A central tenet of this thinking is management by numbers which helps create a simplified and abstracted view of reality.

Shortcomings of Command-and-Control

When authority flows from top to bottom, management is at risk of making decisions that are not anchored into reality. Insights into real problems and opportunities become obscured by this simplification and abstraction of information.

When bottom-up communication is reduced to one-line messages and green/yellow/red progress reports, it reduces the number of interactions creating even more distance between the few "in command" and the reality.

Command-and-control thinking is not an effective way of aligning autonomous teams because top-down flawed decisions are likely to clash with autonomous teams. For example, the Spotify engineering culture is waste-repellent: if it works keep it, otherwise dump it. At Spotify they skip or dump handoffs, useless meetings, and corporate nonsense.[18]

In contrast, agile organizations align work with a meaningful purpose … The few on the top provide clear vision, priorities, and missions. Transparency gives a team access to the information and context it needs to make good decisions. Well-informed teams are given empowerment and trust. Access to privileged information is no longer a power source that middle managers leverage to impose their will upon their teams.

Changing the Organizational Model

The shift from command-and-control to agility requires a culture change. In his book "Reinventing Organization"[19] Frédéric Laloux develops a taxonomy of organizational models. The author observes that most modern global corporations are the embodiment of, what he calls, the Orange Organization type where the hierarchical structure dominates. Virtual teams, cross-functional initiatives, and expert staff functions foster the innovative responsiveness that is needed to beat the competition.

[17] Team of Teams: New Rules of Engagement for a Complex World, General Stanley McChrystal, David Silverman, Tantum Collins, Chris Fussell, Portfolio Penguin, 2015.

[18] See https://vimeo.com/94950270.

[19] Reinventing Organizations: A Guide to Creating Organizations Inspired by the Next Stage of Human Consciousness, Frédéric Laloux, Nelson Parker, 2014.

The next stage of evolution, the Green Organization, retains the meritocratic hierarchical structure of Orange but pushes most of decisions down to frontline workers. In Green Organizations *"a strong, shared culture is the glue that keeps empowered organizations from falling apart. Frontline employees are trusted to make the right decisions because they are guided by a number of shared values, rather than by a thick book of rules and policies"*.

Too many enterprises that deploy agile-at-scale lack the empowerment, strong culture, and shared values that are pre-conditions to agile transformation. Enterprise Architects who are used to operating in Orange Organizations are often ill-prepared to drive change toward agility. Architecture governance models that were developed in Orange Organizations get into the way of agile transformation.

John Kotter has developed a model that describes how traditional organizational hierarchies can shift toward this next stage of organizational evolution. For most companies, the hierarchy is the singular operating system at the heart of the enterprise. But the reality is that this system simply is not built for an environment where change has become the norm. Kotter advocates a new system – a second, more agile, network-like structure that operates in concert with the hierarchy to create what he calls a "dual operating system" – one that allows companies to capitalize on rapid-fire strategic challenges and still make their numbers. "Accelerate"[20] (XLR8) vividly illustrates the five core principles underlying a new network system, the eight accelerators that drive it, and how leaders must create urgency in others through role models. Figure 2 compares the two models and shows how they can interlock.

[20] Accelerate: Building Strategic Agility for a Faster-Moving World, John P. Kotter, Harvard Business Review Press, 2014.

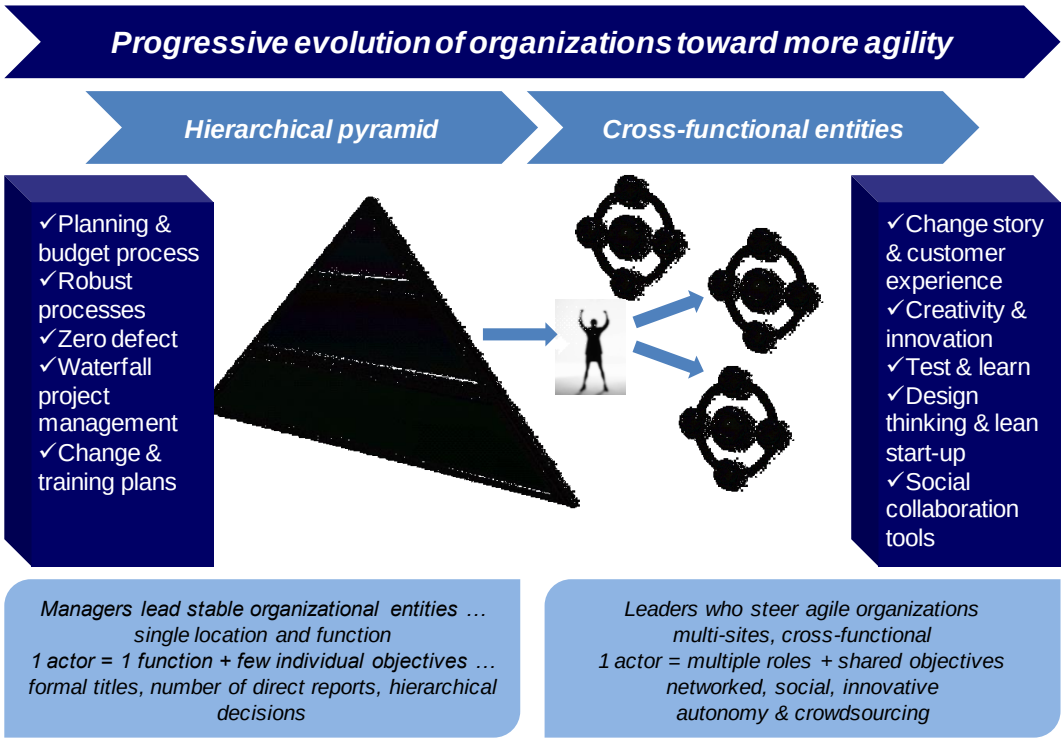

Figure 2: Dual Organizational Model

Let us now illustrate the magnitude of change that is required. General Stanley McChrystal confronted a nimble and agile enemy. To cope with this, he had to change the culture and operating model of an institution that is used to command-and-control thinking:

"We restructured our force from the ground up on principles of extremely transparent information sharing (what we call "shared consciousness") and decentralized decision-making authority ("empowered execution") … We dissolved the barriers – the walls of our silos and the floors of our hierarchies – that had once made us efficient … We looked at the behaviors of our smallest units and found ways to extend them to an organization of thousands, spread across three continents. We became what we called "a team of teams": a large command that captured at scale the traits of agility normally limited to small teams …"

Reflecting on military history, McChrystal attributed the battle of Trafalgar's victory to the organizational culture that Nelson had crafted. This culture rewards individual initiative and critical thinking, as opposed to simple execution of commands. Such a cultural change implies that leaders should not make or approve all important decisions:

"The wait for my approval was not resulting in any better decisions … I came to realize that, in normal cases, I did not add tremendous value, so I changed the process … The risks of acting too slowly were higher than the risks of letting competent people make judgment calls … More important, and more surprising, we found that, even as speed increased and we pushed authority further down, the quality of decisions actually went up …"

If we accept the hypothesis that command-and-control is not an effective alignment approach to steer an agile organization, what is the alternative? Leaders at the top need to provide guidance, feedback, and support to their teams. They need to lead with purpose, which requires strategic clarity.

Business Architecture Patterns

In a seminal paper,[21] Michael E. Porter writes: *"The essence of strategy is in the activities – choosing to perform activities differently or to perform different activities than rivals."* It all starts with the definition of strategic positions that can be based on customer needs, customer experience, and/or some product or service mix.

Strategy is about creating a unique position that involves a different set of activities that better meet customer needs while delivering superior experience. The way these activities are implemented determines costs (operating model view). The difference between the price customers are willing to pay and costs determines profitability (business model view).

Architecting a business and its corresponding operating model can no longer follow a waterfall process steered in a "top-down" manner. The Lean Startup book[22] has popularized an incremental approach that relies on rapid experimentation and validated learning.

Autonomous teams that are in direct contact with clients are best equipped to define MVPs that are market-tested during rapid learning cycles. Though autonomous teams are free to experiment, they need guidance.

The leadership team needs to define a clear vision which can be translated into a set of missions that are assigned down the organization. The missions are operationalized by agile teams that are empowered to challenge them if needed. The learning process, that lean refers to as catch ball,[23] provides a powerful alignment mechanism if conducted well.

A recent paper from the MIT Center for Information System Research (CISR) illustrates how a combination of alignment mechanisms helped Spotify avoid chaos while protecting teams' autonomy:

- Provide distinct goals and objectives to autonomous teams and align teams without introducing layers of hierarchy
- Set up formal sharing mechanisms that synchronize activities as the number of teams grows
- Define architectural standards that facilitate autonomy by ensuring that individual components are compatible

A new class of technology-enabled business model is transforming industries, the platform. It connects people, organizations, and resources in interactive ecosystems that disrupt incumbents. Airbnb™, Uber™, Alibaba, or Amazon Marketplace epitomize this disruptive power.

[21] What is Strategy?, Michael E. Porter, Harvard Business School Press, 1996.

[22] The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses, Eric Ries, Random House Audio, 2011.

[23] See www.lean.org/lexicon/strategy-deployment.

Traditional business models were built around products or services which were designed on one end of a pipeline and delivered to clients at the other end. When platform-based businesses enter markets dominated by "pipelines", they enjoy a competitive advantage. Why? Because pipelines rely on inefficient gatekeepers to manage the flow of value when platforms promote self-service and direct interactions between participants.

A platform can scale and grow more rapidly and efficiently because the traditional gatekeeper is replaced by signals provided by market participants through a platform that acts as a mediator. Platforms stimulate growth because they expose new supply and unlock new demand. They also use big/fast data and analytics capabilities to create community feedback loops.[24]

Platforms need governance which consists of a set of rules concerning who gets to participate in an ecosystem, how to divide the value, and how to resolve conflicts. Good governance distributes wealth among those who add value in a manner that is perceived as fair. Governance must pay special attention to externalities. For example, Airbnb suffered from new rules issued by public authorities wanting to limit externalities such as its negative impact on apartment rental markets.

Because technology is a key enabler, we will now review the software architecture patterns that make digital business models possible. We will also briefly introduce another type of platform that Michael A. Cusumano designates as "internal platform".[25] They allow their owners to achieve economic gains by reusing or redeploying assets across families of products.

[24] Platform Revolution: How Networked Markets Are Transforming the Economy – and How to Make Them Work for You, Geoffrey G. Parker, Marshall W. Van Alstyne, Sangeet Paul Choudary, W. W. Norton & Company, 2016.

[25] Industry Platforms and Ecosystem Innovation, A. Gawer, M. Cusumano, Journal of Product Innovation Management, 2013.

Software Architecture Patterns

"Software is eating the world", Marc Andreessen.[26]

The rapid evolution of software technology has fueled the growth of digital business. Following Internet giants' lead, some enterprises from the old economy are framing themselves as tech companies; for example, Banco Bilbao Vizcaya Argentaria (BBVA): *"If you want to be a leading bank, you have to be a technology company."*[27]

Internet giants did succeed at retaining the agility of startups while they grow at a fast pace and operate at a global scale. They paid special attention to loose-coupling and team autonomy and they learned how to master distributed computing at scale. Let's illustrate this with Amazon and Google®.

In 2002, Amazon was facing a complexity barrier. The size of its home page reached 800 MB and it took 8 to 12 hours to compile. Jeff Bezos issued a mandate that profoundly changed the way software is created and the enterprise is organized. Steve Yegge has reported this in a post.[28]

1. *"All teams will henceforth expose their data and functionality through service interfaces.*
2. *Teams must communicate with each other through these interfaces.*
3. *There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.*
4. *It doesn't matter what technology they use. HTTP, CORBA, Pub/Sub, custom protocols – doesn't matter. Bezos doesn't care.*
5. *All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.*
6. *Anyone who doesn't do this will be fired.*
7. *Thank you; have a nice day!"*

By shifting toward modularity and APIs, Amazon became well positioned to open its distribution and logistics capabilities to third-party vendors. The self-service nature of the platform made it easy for vendors to sell and distribute their products in a frictionless manner. This helped Amazon compete against eBay leveraging a business model which is different.

[26] See www.wsj.com/articles/SB10001424053111903480904576512250915629460.

[27] See www.bbva.com/en/want-leading-bank-technology-company.

[28] See http://homepages.dcc.ufmg.br/~mtov/pmcc/modularization.pdf and https://plus.google.com/+RipRowan/posts/eVeouesvaVX.

In 2004, Jeffrey Dean and Sanjay Ghemawat from Google published a paper[29] that described the MapReduce programming model. This innovation deeply influenced distributed computing. MapReduce is based on a functional style that makes it easy to automatically parallelize and execute code on large clusters of commodity machines.

It allows developers without any experience with parallel and distributed computing to easily utilize the resources of a large distributed system. It is also highly scalable and resilient to hardware or network failures.

Two years later, a team from Google published a paper[30] that describes a distributed storage system for managing structured data designed to scale to a very large size: petabytes of data across thousands of commodity servers. This system exploits immutability which simplifies concurrency control: *"we do not need any synchronization of accesses to the file system when reading from SSTables. As a result, concurrency control over rows can be implemented very efficiently"*.

The vast amount of data Internet giants are gathering is best exploited using Artificial Intelligence (AI). Analytics algorithms help predict customer behavior and recommend products. Deep learning technology powers new types of applications that leverage computer vision and natural language processing.

The model Internet giants follow is based on developing many custom solutions to support their own products and services. They document many of these internal solutions in white papers that later evolve into open source projects. The vast majority of leading-edge technology is freely available, including advanced AI libraries such as TensorFlow™.[31] Some of the open source projects even include pre-trained deep learning algorithms that simplify the creation of new AI applications; for example, OpenFace:[32] *"Free and open source face recognition with deep neural networks … Please use responsibly! We do not support the use of this project in applications that violate privacy and security."*

This stream of continuous technology innovation profoundly impacts the way software that supports the enterprise is architected. The rise of automation that ultimately results in the creation of autonomous systems has and will continue to disrupt business and operating models.

We will now focus on a few key software design patterns that should be part of the architect's body of knowledge.

New Rules of Distributed Computing

The design, development, and operation of distributed systems has always been a difficult endeavour. In the past, middleware technology based on transaction monitors and two-phased commit protocols was good enough.[33] Today, it cannot anymore meet the scalability and availability needs of digital operating models.

[29] See https://static.googleusercontent.com/media/research.google.com/fr//archive/mapreduce-osdi04.pdf.
[30] See http://static.googleusercontent.com/media/research.google.com/en/us/archive/bigtable-osdi06.pdf.
[31] See www.tensorflow.org.
[32] See https://cmusatyalab.github.io/openface/.
[33] Essential Guide to Object Monitors, Karen Boucher, Fima Katz, Wiley, 1999.

Architects need to understand and take advantage of the paradigm shift toward new distributed computing models that:

- Decompose systems into distributable parts that run concurrently on commodity hardware
- Are horizontally scalable and elastic to varying workloads
- Take full advantage of modern multi-core processors

Architecting distributed systems is about making trade-offs between operational complexity, performance, availability, and consistency. The CAP theorem states that any networked shared-data system can have at most two of three desirable properties:

- Consistency (C)
- High availability (A)
- Tolerance to network partitions (P)

Splitting a system into distributable parts gives the ability to scale service capacity, using a larger number of shards to serve more users. Replication (data or functionality) in more than one location is required to recover from failures, thus contributing to the high availability quality.

Because traditional concurrent programming is error prone, new "immutable" programming models are important. Reasoning about the possible states of complex objects is difficult. Reasoning about the state of immutable objects is trivial because they can only be in one state and can be shared safely: *"writing correct concurrent programs is primarily about managing access to a shared, mutable state ... If an object's state cannot be modified, these risks and complexities simply go away"*.[34]

Functional Programming (FP) treats computation as the evaluation of mathematical functions. A pure function is a function which given the same inputs, always returns the same output, and has no side-effects. Pure functions are completely independent of outside state and, as such, they are immune to entire classes of bugs that have to do with shared mutable state.

Their independent nature also makes them great candidates for parallel processing across many CPUs, and across entire distributed computing clusters. Because of this, the FP paradigm is used to build large-scale distributed systems and is becoming mainstream. For example, software tools such as Apache Spark™ or Kafka® are written in Scala which is a functional language and languages such as JavaScript or Java® have functional extensions.

Highly distributed computing models create specific challenges on the data side. For example, microservices which can run in parallel on multiple nodes own their data. This makes ensuring data consistency a challenge. The Saga pattern solves this problem.

[34] Java Concurrency in Practice, Brian Goetz, Tim Peierls, Joshua Bloch, Joseph Bowbeer, David Holmes, Doug Lea, |Addison-Wesley Professional, 2006.

New Data Patterns

A Saga is a long-lived transaction[35] that can be written as a sequence of transactions that can be interleaved. All transactions in the sequence complete successfully or compensating transactions are executed to amend a partial execution. Both the concept of Saga and its implementation are relatively simple, but they have the potential to improve performance significantly.

In the past, a clear majority of persistence mechanisms was based on the relational data model. This is changing because the need to scale data processing to levels never achieved before forced Internet giants to invent new types of Data Base Management Systems (DBMS) which rely on different data models; for example, the key value pair or the graph ones.

From a "one size fits all" view of data storage, we are shifting to polyglot persistence. It is now best to use multiple persistence mechanisms, chosen based upon the way data is being used by individual application components. For example, Machine Learning (ML) is best served by persistence mechanisms that are good at handling DataFrames, which is the common data structure used by ML algorithms. JavaScript Object Notation (JSON) is a popular data structure that has often replaced XML. DBMSs such as MongoDB™ which uses JSON documents to store records save programmers the burden of transforming a JSON document into rows and back again from rows into JSON documents.

One of the major reasons architects need to understand new data technology is that it has a direct impact on the ability of the system to meet non-functional requirements. Let's illustrate this with Apache Cassandra®, which implements a sharding algorithm. Partition keys which are defined in the schema drive sharding behavior. If they are not well designed, the sharding algorithm may distribute too much data on one node, which would result in rapid performance degradation. In the old days, this was not a concern because of the physical independence that RDBMS technology provides. Architects who still operate with an old RDBMS/TPC mental model are at risk of architecting non-scalable and fragile systems.

Traditional batch analytics is no longer suited for many of today's business practices. Time-to-insight is long and it often takes several days to get the right data in the right place. The batch model has often remained unchanged even after the big data revolution.

Fast data is overtaking big data. It is characterized by processing large amounts of data coming at high speed that is processed continuously and acted upon in real time. Real-time analytics takes advantage of fast data to produce predictions or prescriptions in a few seconds and sometimes sub-seconds. Let's illustrate the value of fast data and real-time analytics with a few use-cases:

- Detecting what customers are trying to do and proposing help in real-time contributes to a better customer experience
- Detecting fraudulent transactions on-the-fly before they are approved saves money
- Detecting point-of-sale errors so they can be corrected before the client leaves the store improves service quality

[35] SAGAS, Hector Garcaa-Molrna, Kenneth Salem, Department of Computer Science, Princeton University, 1987.

- Pushing special offers based on customers' real-time location in the store generates more revenue
- Analyzing streaming sensor data in real time to monitor patients' health helps lower risks

Technology innovation makes it easier to implement fast data and real-time analytics. Apache Storm™ and Spark paved the way. Apache Flink® takes it to a new level with true real-time stream processing at scale. New business and operating models are enabled by this technology. Even business people need to understand it to architect service systems that deliver superior customer experience and improve operational excellence.

Infrastructure as Code

Software "is eating" the infrastructure world too. Infrastructure as code automates infrastructure using software development practices; for example, source code management with GitHub, Test-Driven Development (TDD), or Continuous Deployment (CD).

The benefits of immutability can benefit infrastructure too. You create and operate your infrastructure using the programming concept of immutability. That is, once you instantiate something, you never change it but replace it with another instance. It doesn't mean that the system never changes, it is just a simpler and more reliable way of managing infrastructure change at scale.[36]

Cloud Native Infrastructure (CNI) is controlled by APIs and managed by software. Running infrastructure with these traits is more efficient and scalable thanks to advanced automation. CNI enables autonomous application management.

A Cloud Native Application (CNA) that runs on a CNI becomes an autonomous system that does not require humans to make decisions. The system notifies a human on an exception basis when it cannot determine automatically what to do.

CNAs need a platform that can pragmatically monitor, gather metrics, and then react when failures occur. Heroku developed a manifesto that describes the rules and guidelines that CNAs should follow. Since then, the original 12 factors have been updated.[37]

In classic environments a server is used to provide everything an application needs, from satisfying the application's dependencies to providing a server in which to host it. Today a build artifact contains everything it needs to run an application component. Container technology encapsulates build artifacts into images that can be easily deployed, accelerating continuous deployment and facilitating cloud portability.

The greater isolation that cloud native computing brings is a key autonomy enabler because it minimizes the dependencies that agile teams must manage.

The cloud is more about the way applications are developed than the infrastructure that hosts them. The traditional boundary that separates application from infrastructure is blurring. The principle *"you build it, you run it"* promotes cross-functional DevOps teams and busts traditional IT silos. This evolution impacts

[36] Immutable Infrastructure: Considerations for the Cloud and Distributed Systems, Josha Stella, O'Reilly Media, Inc., 2016.

[37] Beyond the Twelve-Factor App, Kevin Hoffman, O'Reilly Media, Inc., 2016.

architecture roles and the architecture process which needs to be better interlocked with engineering processes.

The Agile Architecture Framework (AAF)

Because of the disruptive nature of the trends we have described in the paper, we believe enterprises need a new architecture framework. Let us describe the changes that will impact architecture roles and the architecture process.

Architecture Roles

"Enterprise Architecture is finally working for us, not against us", Target IT VP.

A paper published by IEEE Software describes the evolution of the software architect's role in the digital age.[38] Table 1 summarizes the key tenets that result from analyzing this paper:

Table 1: The Architect's Role in the Digital Age

1. Software teams are increasingly embracing tools and practices that help them avoid, decouple, or break down big, up-front decisions.	6. Modern software architecture … also emphasizes design for automated deployment, dynamic scaling, automated failover, predictive monitoring, and many other advanced runtime considerations.
2. Evidence from the field suggests that most successful architectures and their decisions are created through a collaborative team effort, rather than relying only on architects.	7. New architectures and approaches, such as the cloud and DevOps, which help traditional companies compete with digital disruptors, necessitate changes to organizational structures and working cultures (inverse Conway's law).
3. Instead of architectural decisions being documented in stacks of binders, Internet-scale companies' architectures live in the code.	8. Architecture decisions do not over-constrain detailed design: Loosely-coupled systems and DDD's evolving order principle.
4. Internet architectures blur system context boundaries, replacing single software applications with interconnected ecosystems of services in an API economy … making it difficult for teams to freeze designs, rendering design flexibility a top architectural quality.	9. Architects must also transport and combine knowledge from what used to be isolated domains, such as embedded systems, analytics, infrastructure design into mainstream software development teams, playing a horizontal connector role.
5. Deploying software into a connected world and onto a variety of devices elevates architectural concerns previously confined to specialized domains: cybersecurity, power consumption, runtime configuration, and automation to name a few.	10. Times of rapid change require architects who can act as mentors and bridge builders among project teams, across domains, and between different layers of the organization.

In a nutshell, architects are becoming mentors and bridge builders. The architect's role is becoming more demanding because concerns that were previously confined to specialized domains need to be connected. The

[38] See www.computer.org/csdl/mags/so/2016/06/mso2016060030.pdf.

successful architect is a T-shaped individual who can connect the dots and has enough depth in a few domains to remain credible *vis à vis* engineering teams.

In a recent talk, Target's VP of Architecture Joel Crabb says that *"the big architecture era is over"*.[39] He reports asking search companies to find if digital native companies have Enterprise Architects he could hire. He discovered that Netflix® and Facebook don't have Enterprise Architects, and though Amazon is hiring a lot of architects their role is about helping clients move to Amazon Web Services (AWS).

Joel Crabb claims that Enterprise Architecture is being disintermediated. Specialty or domain architects form an unnecessary layer that sits between agile teams and other roles. For example, what's the value of a business architect when product managers that lead agile teams are coming from the business and are empowered to make true business decisions?

Does it mean that Enterprise Architecture is dead? Joel Crabb does not think so. He develops the model of an "architecture supported" engineering culture. To stay relevant, architecture teams ensure engineering teams can quickly take advantage of great ideas by:

- Providing an enterprise-wide architecture vision that seats on a page
- Building an engineering culture with social coding and open repos (e.g., GitHub)
- Letting engineers write standards and ask architects to facilitate the process
- Automatically measuring compliance
- Managing right at the edge of chaos

On the enterprise-wide architecture vision, Joel Crabb developed one that is based on an internal software platform that promotes reuse and standardization. The architecture model distinguishes the platform from the tenants. Tenants interact with the platform using an event-based asynchronous communication mechanism.

Eventual consistency is the norm unless a specific business need justifies an exception to the rule. The resulting system is very robust, can scale up and down, and is highly resilient to all types of failures. The reference architecture model can be represented in a simple diagram that fits one page.

This type of internal platform requires governance too. Target's architecture governance tenets are summarized below:

- Every system has a context: Platform or Tenant
- Every system has a defined scope: Data, Process, Logic, Aggregation, UI
- Tenants are decoupled from other tenants and the platform
- Context decides the required amount of enterprise governance

[39] See www.safaribooksonline.com/library/view/oreilly-software-architecture/9781492028116/video318616.html.

On managing at the edge of chaos when a hundred squads move very fast, Enterprise Architects should avoid stifling innovation and must become comfortable with change. Because engineering moves faster than you can track, ideas spin up and shut down in weeks and technologies are rapidly adopted and discarded, management and Enterprise Architects must let go.

At Target, the architecture's purpose is to create the environment where great ideas win by getting them to production quickly. For example, the architecture team helped Target reduce the number of pricing systems from seven to one. From an architecture team viewpoint, going from seven to one pricing system is a big achievement. The engineers are doing it because they think it is a good idea, not because the architecture team told them to do so.

Architecture Process

The old paradigm of big up-front design is replaced by the notion of Minimum Viable Architecture (MVA). The MVA defines the minimum set of architecture decisions and infrastructure capabilities that condition the start of the first (or next) agile iteration.

Much has been written on MVA[40] and many variables influence its definition; for example, requirements instability, unknowns, tolerance to risk, cost and system's evolvability.

The last variable refers to the ability of a system to respond to changes that are not yet known. The design strategies listed in Table 2 can help architect a system that can evolve more gracefully.

Table 2: Evolvability Design Strategies

Design Strategy	Description	Techniques
Separation of concerns	Separating a software system into distinct solutions, such that each section addresses a separate concern	Aspect-oriented programming, clean architecture
Modularization	Decomposing a system into modules driven by information hiding and separation of concerns	Domain-Driven Design, Design Structure Matrix
Delay design decisions	Making design decisions only when facts are known, instead of over-compensating for unknown requirements	Set-based concurrent engineering
Meta-modeling	Modeling the concepts and relationships of a modeling language/notation	Meta-object protocol

[40] For example, see Minimum Viable Architecture – Good Enough is Good Enough in an Enterprise, James Governor, 2017 (http://redmonk.com/jgovernor/2017/06/13/minimum-viable-architecture-good-enough-is-good-enough-in-an-enterprise/) and How to Create a Minimum Viable Architecture, Deepak Karanth, 2016 (https://dzone.com/articles/minimum-viable-architecture).

Design Strategy	Description	Techniques
Augmented intelligence/ knowledge-based systems	Knowledge an algorithm can leverage to lower the cost of changing the behavior of a system	Rule Engine, Machine Learning, Deep Machine Learning

Each design strategy has strengths and weaknesses. For example, using a rule engine will make it easier to evolve business rules. On the other hand, it is likely to create more coupling because of all the interactions of the rule engine has with the rest of the system.

Traditionally architects make the decisions that are difficult and costly to change. By making evolvability an explicit architecture objective, the difficult to change decisions should be fewer. However, some decisions will remain difficult to change. Therefore, it is important to single out those.

Suudhan Rangarajan explains how Netflix uses the Amazon decision taxonomy to classify decisions into type 1 and type 2.[41]

Type 1 decisions are consequential and irreversible or nearly irreversible – one-way doors. Type 1 decisions must be made methodically, carefully, slowly with great deliberation and consultation.

"Most decisions are not like that. They are changeable and reversible, they are two-way doors. If you've made a sub-optimal type 2 decision, you don't have to live with the consequences for that long. Type 2 decisions can and should be made quickly by high judgment individuals or small groups." (Jeff Besos)

Type 2 decisions can be made by autonomous agile teams, while Type 1 decisions require governance.

At Netflix there are three categories of type 1 decision: shared libraries and communication, synchronous *versus* asynchronous, and data architecture. Type 1 decisions are contingent and depend on the context of the enterprise; for example, its business and technology strategies, or its maturity level.

We believe this classification matters for architecture governance. Enterprise Architects can no longer wait for agile teams to request their stamp of approval, they must:

- Develop an overall architecture vision that helps guide engineering teams
- Identify type 1 decisions they instruct on their own initiative in a pro-active way

The architecture process should move away from a waterfall/gated style. We need to define a comprehensive end-state. This is what motivates the development of a new Agile Architecture Framework (AAF). We invite the community to contribute to this initiative. We propose to organize its development by Epics as described in Figure 3.

Because the journey toward that end-state should be incremental, we plan to develop an agile architecture maturity model which has the ambition of helping enterprises succeed this transformation. It will clearly articulate pre-conditions and success factors when moving from one maturity level to the next.

[41] See https://www.safaribooksonline.com/library/view/oreilly-software-architecture/9781492028116/video318619.html.

AAF Development Approach

We propose to structure the development of the AAF along three themes:

- Autonomy, isolation, and alignment
- Architecture process and roles
- Architecture body of knowledge

For each of the themes, we have identified a set of development Epics (see below).

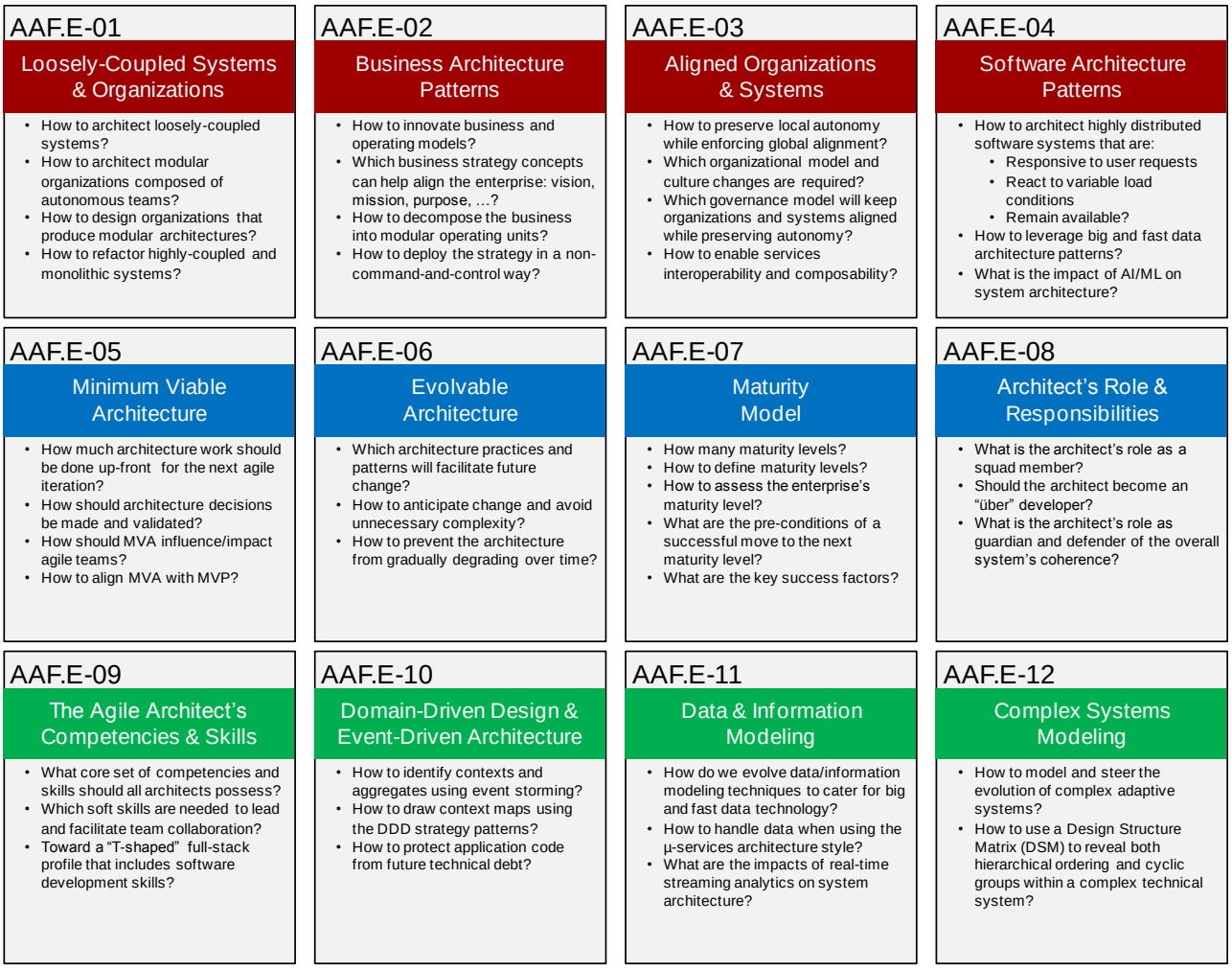

Figure 3: AAF Development Epics

About the Authors

Hervé Barbazange is group Enterprise Architect at Société Générale. He holds a degree in engineering (Centrale Paris) and a Master's degree in Enterprise Architecture.

Dr. Peter Beijer is a recognized pioneer in DXC's architecture methods with leadership roles in the architecture profession for many years. He serves on The Open Group Governing Board and chairs the Professions Program. He received a PhD on economies of meaning in innovation processes.

Jean-Marc Bunouf who was Information System Architect at MMA Insurance is now group Enterprise Architect at Société Générale. He holds a degree in computer engineering (UTC).

Carl Kinson is DXC's Chief Architect and honored Distinguished Architect. He is a technology thought leader with over 20 years' experience of designing, building, and managing complex solutions for clients, with a strong focus on embracing new technologies and skills to develop businesses and individuals.

Frédéric Lé is Technology Strategist working for DXC's Corporate Technology Office. He leads the development of DXC's new Agile Architecture Framework. Frédéric graduated from Audencia Business School and holds a post-graduate degree in computer sciences from Paris Dauphine University.

Jean-Pierre Le Cam is group Change Management expert at Société Générale. He holds a degree in engineering (Télécom Paris), and is change master at the ESSEC Change Management academy.

Antoine Lonjon is Chief Innovation Officer at MEGA International. Antoine has contributed to the foundation of the HOPEX Enterprise Architecture toolset and is also involved in standards organizations, such as The Open Group and Object Management Group (OMG).

Jérôme Régnier, who was Enterprise Architect for the French Ministry of Social Affairs and Health, is now group Enterprise Architect at Societe Générale. He holds a degree in computer engineering (ENSIIE – Mines-Telecom Institut).

About The Open Group

The Open Group is a global consortium that enables the achievement of business objectives through technology standards. Our diverse membership of more than 600 organizations includes customers, systems and solutions suppliers, tools vendors, integrators, academics, and consultants across multiple industries.

The Open Group aims to:

- Capture, understand, and address current and emerging requirements, establish policies, and share best practices
- Facilitate interoperability, develop consensus, and evolve and integrate specifications and open source technologies
- Operate the industry's premier certification service

Further information on The Open Group is available at www.opengroup.org.