Survive & Thrive in Agile: A Guide for Enterprise Architects
(With TOGAF & Archimate)

V2.0

Contents

Introduction

The goal of this guide is *to enable enterprise architects to deliver greater business value in Agile environments*. In order to achieve this, it will do two things. First, it will demonstrate that bringing together Enterprise Architecture and Agile is not only possible, but in fact very beneficial for both camps and indeed the business as a whole. Second, it will provide advice on how to integrate the two in practice, using real-world use cases and examples to give practitioners the best chance at success. In other words, we will prove it can be done and then we will show you how to do it. Without further ado, let us get into it.

Part 1: Do EA and Agile Mix?

Why is a guide necessary in the first place? Let us explore this apparently antagonistic relationship between Enterprise Architecture and Agile.

A False Contest

For most of its history, EA has been a relatively slow, monolithic activity that took a long view of the future and exhaustively proceeded to plan about change. This sluggishness coupled with architects' lack of communication skills and their failing to adopt a sense of urgency when it came to the business' needs (the famous Ivory Tower Syndrome) left EA with a battered image. The fact that for a long time there were no mature tools to support professionals did not help either. Of course, current EA best practices steer away from this passivity, but poorly managed individual architecture practices together with the stigma of the past means the image is hard to shake off.

The Agile movement, on the other hand, embraces speed, responsiveness, as well as a fail-forward approach to resolving issues. An agile business is one that focuses on adapting to change. Here are the actual values and principles from the Agile Manifesto that that underpin this mindset as a reference:

Values

1. Individuals and interactions over processes and tools.
2. Working software over comprehensive documentation.
3. Customer collaboration over contract negotiation.
4. Responding to change over following a plan.

Principles

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity – the art of maximizing the amount of work not done – is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

As evidenced by these, Agile promotes Collaboration, Practicality, Responsiveness, Transparency, Autonomy, and Trust in order to ensure the organization is capable of withstanding change. These features together produce *Agility*. Agile rejects the waterfall model and what is known as Big Design Up Front, as well as comprehensive documentation and long-term planning.

So, the problem of associating Enterprise Architecture and Agile stems from a falsely perceived incompatibility between the two practices that originates in EA's historical shortcomings and the negative image they created. An image of a slow, lumbering practice, in complete antithesis with the Agile ethos and therefore presumably incapable of delivering any value in an Agile context. EA is perceived as slow and "cautious", while Agile is fast and "confident". One is the past; the other the future. Or at least that is what many people seem to think as they fail to notice their complementarity.

We, however, think differently. First, we acknowledge that EA is indeed more calculated and meticulous in nature, and therefore it "moves" at a slower pace. However, that is simply because it is different than Agile, not inferior. Enterprise Architecture serves a different purpose, uses different means and tools, requires different skills from its practitioners. We will expand on this as we go forward.

All Speed, No Direction

Change has historically been a major problem for the enterprise, and the larger the enterprise the more difficult it found it to adapt to changing market conditions. A big part of the responsibility for that lies with the comprehensive way in which change was dealt with – lots of detailed documentation, initiatives spread across long time windows, rigid processes.

Meanwhile, Agile advocates taking small steps very often in order to be able to change direction at a moment's notice if necessary. The reason this mindset propelled it to the top and turned it into the hot trend that it currently is, is that Agile approaches work really well in a world made

increasingly volatile by technology breakthroughs, changing social trends, ever-changing regulation etc. They greatly improve responsiveness to change. Yet, they are not a catchall recipe for success.

Consider an Agile-only environment. In smaller organizations, a few agile/DevOps teams can coordinate change amongst themselves, and the lines to management are short enough that strategic direction can be conveyed to teams directly. However, the larger an organization becomes the more interconnected and interdependent its components are (capabilities, resources, processes, systems). This complex landscape of dependencies is the reason Enterprise Architecture is vital in aligning disparate parts of the enterprise with the overall strategic direction, thus ensuring adaptivity.

Massive enterprises may have hundreds of agile teams, each working on their own little project, unaware of the bigger picture, which is why more coordination is needed. Team-level product owners often lack this insight, which makes it likely for them to make decisions that have adverse consequences at the enterprise level. Similarly, if agile teams disregard their environment and proceed to build agile silos the result will not be a flexible organization. The lack of awareness – and therefore coordination – ultimately makes future change even more difficult. When applied inadequately, Agile approaches can actually harm the overall adaptivity of the enterprise.

Good architecture is essential in this case. Of course, the days of modeling the entire enterprise in the hope of being able to answer any question are long gone. Modeling all current and future states together with their associated roadmaps and manually keeping everything up to date – especially in the new paradigm of small and frequent changes – is too much overhead. However, the undeniable fact is that without architecture you are just on a fast track to nowhere. As the Cheshire Cat puts it in *Alice in Wonderland*, "If you don't know where you're going, any road will take you there".

Use Cases for EA & Agile

Here are some use cases for combining enterprise Architecture and Agile. They illustrate how this integration can bring value to the organization.

Prioritize on Business Value

Based on an analysis of your enterprise architecture, you can see how different epics and features are related to business processes, capabilities, business goals, and associated stakeholders. Tracing through your architecture models, you can find out which business goals or stakeholders a feature supports and decide whether that is more important than other features. Objective information like this may help a product owner put a stop to decibel-driven prioritization, for instance, where the users that complain the loudest get their wishes granted.

Coordinate Work

Identified architectural dependencies can be used in planning agile activities. For example, if Feature A depends on Feature B, build feature B first. This is all the more important in what SAFe calls enablers – aspects of the solution that don't provide functionality directly to users, but support functionality indirectly and enable successful long-term evolution. The framework's architecture runway concept is an example of this, where you lay down the technical foundation for new features just in time. This concept can be extended beyond software, though. Investing in, say, staff skills, may provide a foundation for many future developments and improvements.

Track Progress

If some feature or enabler is delayed, e.g. by an agile team wanting to push it to the next sprint, the impact on the overall solution needs to be factored into that decision. There may be good reasons for this team to prioritize something else, but often they do not have enough information to understand the consequences. Being able to trace back to the overall goal will demonstrate how a decision will affect a release, a project completion date etc. It also helps teams answer questions like: *How does this impact the organization's business goals? Which customer groups will be affected? Can we mitigate these effects by prioritizing other features, possibly of other teams*?

That is where architecture models are of great help, especially to enablers. Since enablers do not provide end-user functionality, few people will complain when you postpone them. However, they are often important for many different features and for the solution's long-term viability, so keeping track of these dependencies is key.

Update the Architecture

In an Agile context, the architecture is not some static description of a long-term future state. Rather, it is a shared instrument that provides vision, gives direction, and ensures coherence across the enterprise, based on both top-down and bottom-up inputs. Business strategy, goals, and desired outcomes provide the top-down direction, whereas local innovation and improvement offer bottom-up change. Thus, various stakeholder groups in the enterprise can work together on the architecture, adding new insights from their own subject areas and perspectives.

Use Cases for EA & DevOps

Having seen some valuable applications of enterprise architecture within the realm of Agile, let us now also consider DevOps in relation to EA. Here are a few main use cases.

Correlation & Propagation of Events

Making sense of vast organizational landscapes is difficult, or nearly impossible without the help of architecture. Let us say you need to explore how these security events, i.e. social engineering, phishing, viruses, and physical intrusion, propagate through the architecture to come up with a sensible risk assessment and recommendations.

How would you know what is related to what, without a big picture view of the organization and its components? You could certainly implement some security measures in a hurry and tell the board of directors you are taking solid action to secure the organization, but that would be the equivalent of swinging a sword in the dark. You might keep the bad guy at bay, or (more likely) you might do nothing at all. DevOps can surely benefit from architects' input.

Finding the Cause of a Problem

Identifying problems as part of optimization initiatives is significantly easier when dealing with more abstract representations. EA is nothing if not a good example of such a representation. As such, it supports valuable methods and techniques for eliciting existing or potential problems thanks to things such as root cause analysis. This approach works well even when dealing with a complex network of systems.

Business Continuity

Finally, EA can help DevOps ensure business continuity. That is because it helps answer questions such as *What would be the potential impact of a change?* Or *What can go wrong if we change this system, and how do we mitigate against that?* By highlighting key dependencies between the various aspects of the enterprise (both within but also cross-domain, e.g. how a technology infrastructure element supports a key business capability) and enabling scenario and impact analyses, architecture is a reliable source of valuable information for DevOps departments.

Part 2: How Do I Integrate EA and Agile?

Part 1 communicated the point that Enterprise Architecture and Agile are compatible, and that businesses stand to benefit significantly by providing an EA foundation to their Agile practices. Now let us go over what the role of the enterprise architect is in an Agile context, and also look at what an EA-Agile integration might look like in practice using architecture models and appropriate tool support.

Bridging the Gap

Approaches such as the Scaled Agile Framework (SAFe), Disciplined Agile Delivery (DAD), and Large-Scale Scrum (LeSS) were developed precisely to enable coordinating teams in the context of large organizations. Still, we find their approach to architecture lacking. While they are a step in the right direction, they are also quite focused on the needs of software development, which makes their perspective heavily skewed towards software and solutions. Fortunately, there is more to an enterprise than software.

Take SAFe, for instance (see Figure 1). It uses a layered, iterative approach where we find the agile teams in the bottom layer. These deliver their results in a typical agile frequency of 2-3 weeks. In the next level up, the results of these teams are integrated and released using solution architecture concepts like the Architecture Runway and the Agile Release Train, which ensure that these fit together. This Program layer iterates at a speed that is a few multiples of the Team level, so it has shippable products every 2-3 months.

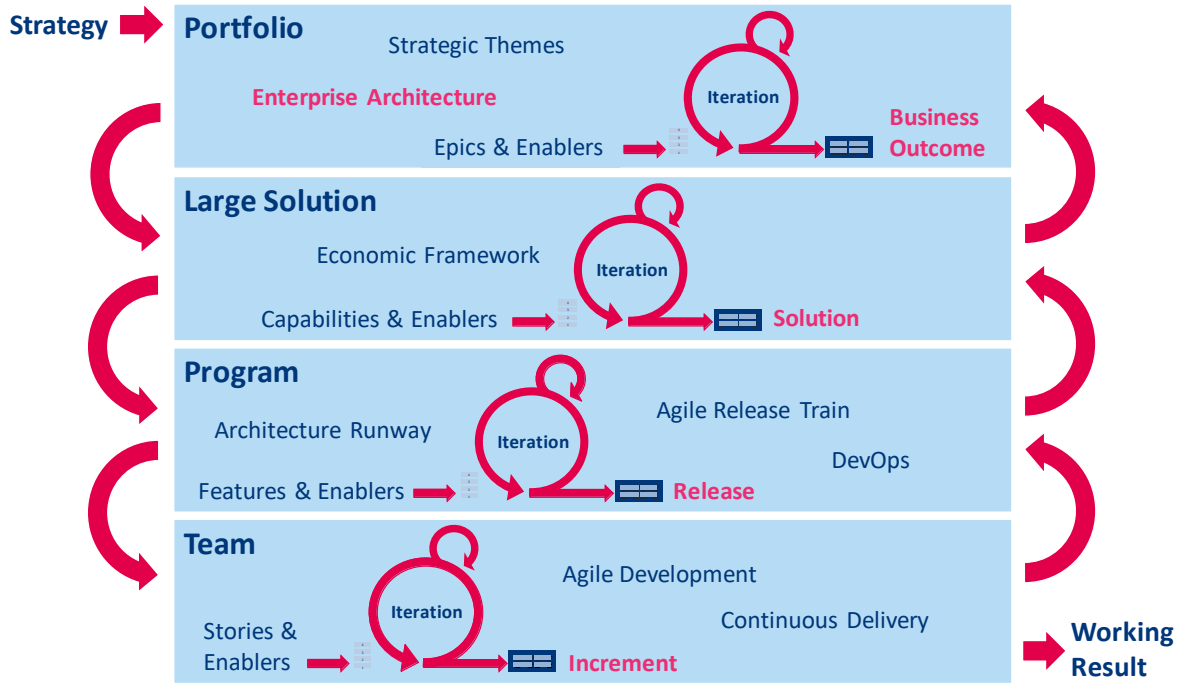

Figure 1: Scaled Agile Framework's layered, iterative approach

The Large Solution level takes care of large, complex solutions spanning multiple Agile Release Trains and suppliers, addresses regulatory compliance, standards, and other external constraints, and provides financial boundaries to guide decision making.

Finally, the Portfolio level at the top is where large, longer-term developments are positioned. This is also where Enterprise Architecture can finally get an entry point and find its own place. The business strategy feeds into this layer and provides the context for large-scale, high-impact architecture decisions, priority setting and budget allocation.

This top layer is where enterprise architects might look to use methods such as TOGAF to try and create value. TOGAF also has an iterative structure, shown by the familiar crop circle diagram of its Architecture Development Method (ADM). However, the iterations in the ADM do not make this into an agile architecture approach, since true agility needs full end-to-end iteration from idea to working solution and back.

Applying TOGAF in an Agile context will therefore require considerable adaptation. First of all, architects will need to become more outward-looking and consequently more business oriented when building and managing their architecture, e.g. pay attention to the end customers and the desired business outcomes. A business outcome could be an end-customer product, which is described by services, capabilities, software products and other artifacts, but also a business transformation that implements a new strategy. Just remember to stick to a "just in time, just enough" type of architecture.

Moreover, instead of first having one or more architecture iterations in the ADM and only then addressing the realization of that architecture and 'governing' the implementation in Phase G, the phases of the ADM should be considered as running in parallel, where each provides meaningful results in a regular cadence, just like other activities in an agile context.

Figure 2 gives an impression of this. At the top we see the development and evolution of the enterprise vision. Next, the baseline architectures are updated continuously with information on the current state, as provided by agile and DevOps teams. In an ideal situation, this information is taken directly from tools like CMDBs that monitor the operational environment or from process mining tools, for example.

The next level down shows the evolution of the target architectures. This is where enterprise and other architects spend most of their effort. Finally, at the bottom level the architects guide and govern change in continuous interaction with the agile teams. This is not just a one-way street where architects give direction to developers.

Closing the feedback loop is essential in providing customer feedback, injecting innovative ideas on new technology, and in general adjusting architectures and prioritizing developments in changing circumstances in order to optimize the business value created. This short feedback loop is *the* critical success factor of agile approaches, and architects should learn from that.

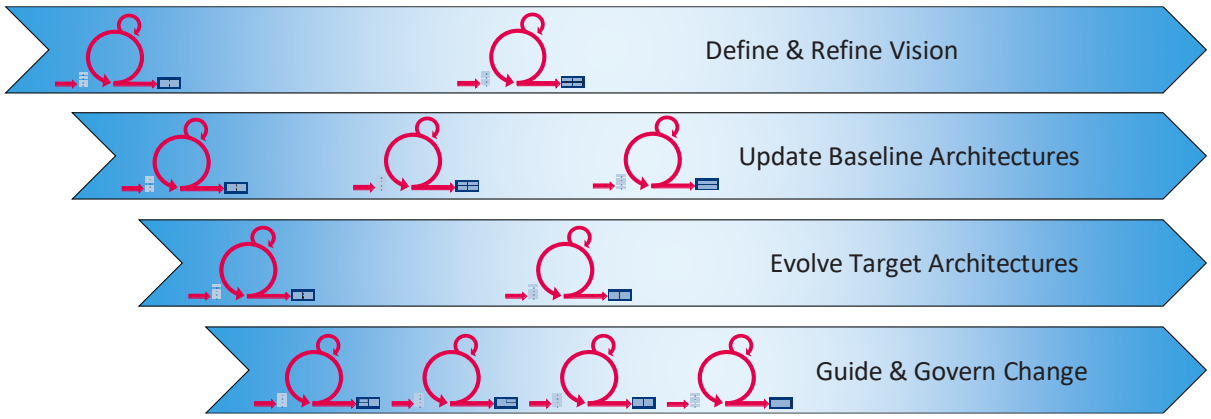

Figure 2: Transforming the TOGAF ADM into a value stream-oriented architecture practice

TOGAF's document-heavy approach also does not sit well with Agile. Rather than documents, the various architecture activities should provide a continuous stream of value in the form of architecture *decisions*. These decisions serve both as input to other phases and to guide the implementation. Some decisions may be conveyed directly to the agile teams by architects; others will need to be recorded in a way that makes them accessible to anyone involved. Rather than extensive documentation that is difficult to search or analyze, we think that architecture *models* are a much better way to record those decisions.

Finally, the key value architecture brings to the table is that it can enhance the agility of the solutions you are developing. That is, you need to ensure not just the agility of the development process, but also the agility of the end result you are building (within the organizational context).

After all, what follows after the software development process arguably matters most, since that's how a solution will be spending the vast majority of its lifecycle – as a finished product, hopefully creating value for the business. There's little point developing great software the right way if it then gets released in a slow-moving, overly complex, and inefficient organization. In the end, it is not going to make a difference – the enterprise moves at the speed of its slowest moving part.

But Agile software development mainly focuses on responding quickly to changing requirements; it doesn't really address the agility of the resulting products. Often, there is the underlying assumption that software is 'by definition' agile since you can always change it, e.g. by refactoring. Although software may be more agile than constructs in the physical world, this assumption is naïve at best and potentially dangerous.

As Grady Booch once defined it, architecture considers those design decisions that are costly to change. For example, some years ago BiZZdesign changed its platform architecture from a centralized to a distributed model. That is the type of architectural decision that has a deep impact and cannot be taken lightly, since it would take a very substantial effort to change it again. A solid architecture practice helps you identify such decisions, avoid mistakes and promote solutions that are easy to adapt in the areas where change is expected. Let us expand on this.

System Agility Is Key to the Mix

This other type of agility we call system agility. We define system agility as having organizational and technical systems that are easy to reconfigure, adapt and extend when the need arises. An agile system is one that simplifies such actions as:

Making changes

How easy is it to make some change to the organizational and/or technical system? Some relevant aspects are customizability (by users), adaptability (by system management), analyzability (by designers), changeability (by developers), scalability (e.g. to accommodate higher volumes). To ensure these features are contributed towards effectively, there are a number of good practices architects ought to follow.

The most important of these good practices are:

- supporting the separation of concerns (i.e., use layering and modularization to concentrate functionality in specific places, which makes changes as local as possible)
- ensuring low coupling and high cohesion (i.e., a low number of relations between subsystems and a high internal cohesion, which simplifies analysis and avoids changes being propagated throughout the whole system)
- and, finally, encapsulation (i.e., creating clear interfaces and allowing no access from environment to internal implementation, the big strength of which is enabling the implementation to be modified without affecting the environment)

Deploying changes

This is particularly important in the context of Agile and DevOps processes, where changes are deployed often, and feedback is collected early. The key aspects for architects to keep in mind here are:

- Learnability: How easy is it for different stakeholder groups to learn to work with new systems, procedures etc.?
- Installability: How easy is it to roll out a new system?
- Testability: How easy is it to test the system, particularly when compared to other existing systems?
- Manageability: How easy is it to manage, control and maintain the system?

Taking these variables into account and making them a part of your system assessments is bound to improve your ability to deploy new systems that support agility, instead of detracting from it.

Dealing with the consequences of changes

This simply means that if something goes wrong during a change, the effects should be minimal and easily corrected to minimize the risk of disruption to day-to-day operations. Fault tolerance, redundancy, and simple roll-back mechanisms facilitate this.

Integrating solutions

Say a new or changed business process, information system, component, or other element needs to be connected with the rest of the landscape. This requires easy interoperability. How does EA make its mark here? The answer is by evangelizing and enforcing relevant standards across the enterprise. We will get back to this in a little bit, but for now let us push forward.

Decoupling solutions

Ease of decoupling is important because an agile system should prevent local changes from propagating unduly throughout. Also, reusability is fostered if elements can easily be lifted out of their context and plugged in elsewhere, i.e., there is no unnecessary dependence on the environment. The organization should aim to design components for independence and reuse, which requires an architecture backbone that provides context for reuse – thus enterprise architecture enters the scene.

So, what is the biggest enemy to system agility? Simply put, *complexity* – having many relations between the elements of a system. This is because complexity takes a long time to grasp (which is necessary to make informed decisions), changes themselves are a lot of work, local changes may have many unforeseen side-effects (sometimes in very unexpected places), and finally because it forces you to do lots of testing. To deal with the challenges of complexity, we recommend two practices – partitioning and alignment.

Partitioning is a principal strategy to combat complexity. Try to create autonomous subdivisions of your architecture and have clear responsibilities for each of them. As you do this, focus on synergistic relationships whereby 'If A needs B *and* B needs A, they belong in the same partition, because they will always be used together'. However, if that is not the case, they are autonomous, and you can put them in separate partitions. Work on striking a balance between controlling complexity *inside* systems and the need for interoperability *between* systems.

Partitioning also simplifies business alignment, since it also reduces the complexity of the decision process, i.e., having many people/organizational units who have something to say about a system. A complicated decision process is often the direct result of a lack of alignment between business capabilities and the underlying IT systems. For instance, consider having a monolithic system that needs to serve both a commercial front-office department (focused on quickly creating new product offerings) as well as an administrative back-office (focused on stability).

So, to ensure alignment, architects should aim to create autonomous business capabilities (using their knowledge of the big enterprise picture) that are responsible for their own processes, systems and data.

Standardization

Now let us take a moment to address the subject of standardization, which we briefly touched upon before, as finding it in an Agile guide is bound to have raised a few eyebrows, without a

doubt. Use of standards is one of the means by which enterprise architecture ensures different pieces of the (organizational) puzzle play nicely together, thus delivering alignment and efficiencies.

In the case of Agile development and integrating solutions, the clarity and structure made possible by standardization cannot be overstated. Of course, at first sight standardization and agility seem quite contradictory. After all, standards are rigid things, right? Well, the answer is basically yes, they are inflexible, but – and this is the catch – that is actually not a bad thing for the purpose at hand.

When applied the right way, use of standards augments flexibility and agility because by specifying the 'rules of the game', it makes it easier to function within that well-defined 'playing field'. In the case of software, this means developing solutions with standardized interfaces that combine and can be reused effortlessly. Think of power plugs or Lego bricks, for instance. So, if there is 'good' standardization but also 'bad' standardization, how do you make sure you do it the right way?

Well, not accounting for the industry you are in, or the specific changes you are trying to carry out, two types of standardization stand out, namely lengthwise (good) and crosswise (bad). Let us have a quick look at both.

Lengthwise Standardization

In the case of lengthwise standardization, the same steps in a business process have been implemented for different product lines, services or customer segments, as you can see in Figure 3. This greatly facilitates reuse of a process step for a different product/service/segment, which enhances your agility since reuse supports a higher speed of change.

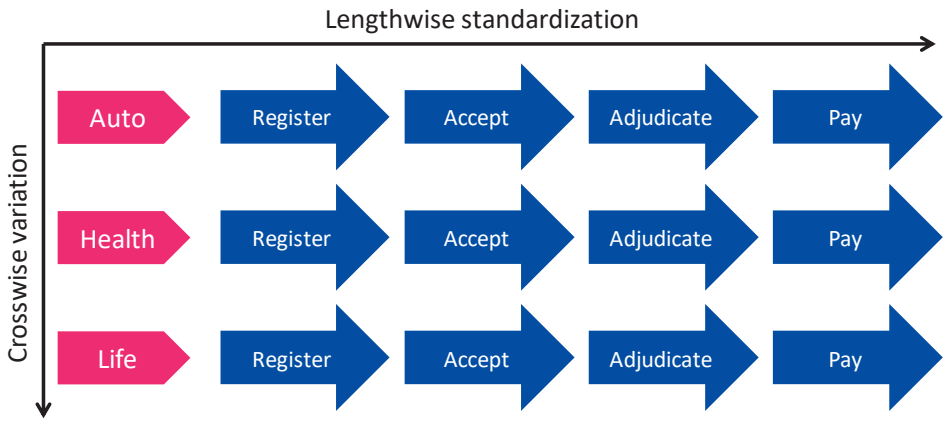

Figure 3: Good Standardization

Crosswise Standardization

Crosswise standardization, on the other hand, refers to using the same implementation within a step for every product, service, or segment (see Figure 4). This is likely to harm your agility because

the result may become too complex and therefore cumbersome to change. When processes from different products, services or segments become interdependent the end result is that change propagates across the organization or system, testing becomes more difficult, and agility suffers.

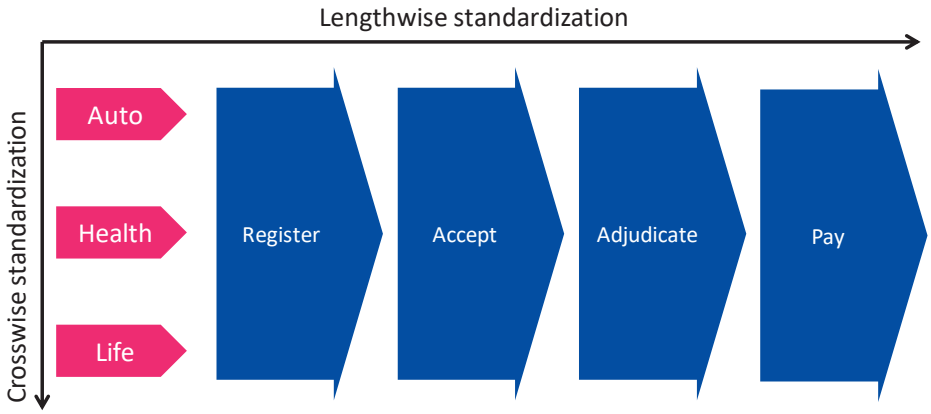

Figure 4: Bad standardization

Enterprise architects should therefore design separate units that can be changed independently. The smaller the unit of change, the easier it is to execute that change. Architecture should facilitate having building blocks that have low complexity (i.e., have few internal dependencies), little or no interdependence with other building blocks, and are easy to test locally.

Additionally, to support creating the right building blocks, you should ideally contain the propagation of changes within organizational units (departments, teams etc.) as much as possible. This means concentrating the knowledge needed for changes and always assigning clear responsibilities.

The Role of Platforms & Models in System Agility

Since we explored the subject of system agility, we also want to mention business platforms, which represent an important basis for system agility. So, what is business infrastructure? Simply put, business infrastructure represents the systems and capabilities that hardly ever change, and as such can serve as a foundation, or infrastructure, for things that tend to change (a lot) more frequently.

To give you an idea of this, consider the electric wiring in your home. That almost never changes throughout your residence's entire lifespan. What you plug into it, however, varies dramatically. Just think of your appliances in 1990, then compare that to the present. The same holds for your company's information systems.

So, for instance, if your data structures are stable but the data itself is changeable, it makes perfect sense to use a DBMS. If, however, your data structures change often, then you are probably better off without one. This is entirely applicable in a business context. Think of your business rules – if

they tend to change often, is it really desirable to have them hardcoded in stored procedures in the database? Probably not.

Naturally, you need to carefully determine *where* exactly you want to be agile. That is because, make no mistake, being extremely flexible in all aspects is neither necessary, nor realistic, and also very expensive. Since everything in your organization and information systems does not change at the same pace, you need to assess the situation and decide according to the types of change you need.

We believe the best way to do this is by leveraging architecture models to perform dependency, impact, business & technical value, and other types of analyses, so you get a clear understanding of the impact of change.

Relying on models is consistent with the general trend in IT towards higher abstraction levels. This is because changes at higher abstraction levels are easier, faster, and less-error prone. So, automating the implementation steps as much as possible is likely to contribute to your organization's agility, as seen in Figure 5. Examples include executable specifications such as BPMN 2.0 process models, automated model transformations, code generation from models, automated testing and test case generation from models, and automated deployment.

If you need agility in your business knowledge (processes, rules, data) your business infrastructure should support this. What you need in this case is an execution environment onto which to run all this business knowledge. By wielding models to identify valuable insights instead of having to resort to code, you stand to benefit significantly. This includes less extensive testing of software changes needed, fewer types of expertise involved, and fewer errors in translation from business-oriented spec (e.g. product rules, laws) to implementation. This is illustrated in Figure 5.

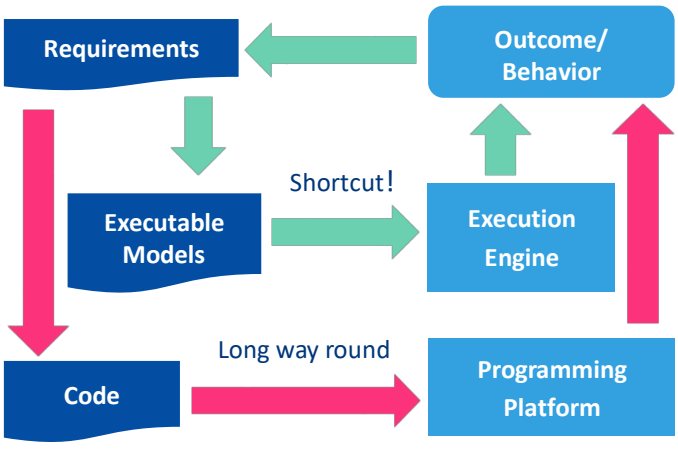

Figure 5: Agility through models

What is more, having such a platform would also fit with the rapid iteration of agile processes. Here are a few examples of what a configurable business platform might look like:

- Business process management and workflow management systems
- Business rule engines
- Low-code development platforms
- Intelligent, configurable Web forms
- Graphical configuration of data models for databases

Bigger Picture = Better Decisions = Higher Agility

Hopefully, that gives you some perspective. Ultimately, this is all meant to help architects get their foot through the door and start thinking about the possibilities. The bottom line is all the agile frameworks we mentioned previously — SAFe, DAD, LeSS – are rather IT-centric. To briefly illustrate this point, the role of the enterprise architect according to SAFe is to "[…] drive holistic technology implementation […]". We believe a true enterprise architect is not focused on technology alone.

Moreover, most organizations do not develop their own software but mostly acquire and integrate COTS packages. Architectural concerns then focus on how well such a package supports relevant business requirements today and in the foreseeable future, and on its integration with the rest of the business and IT landscape. Business architecture is therefore an increasingly important part of the equation.

As such, individual practitioners can play a key role in enabling an Agile practice to flourish by bringing strategy mapping, capability-based planning, value mapping, business process management, Lean Six Sigma, and other business-related disciplines to the table. These offer a valuable opportunity to enhance the picture. A truly agile enterprise needs more than agile IT alone.

Solution architecture can very well be looked after by agile teams, but their work must fit with the overall vision of the organization. For instance, product owners are mostly focused on functional requirements of single solutions, but who takes care of the non-functionals, use of standards, regulatory compliance, or strategic direction? When you consider that, the benefits of EA's big picture view quickly become clear.

EA is able to look outside the software development arena and account for things such as desired business outcomes, capabilities to be developed or improved, resources needed, business processes, IT and physical infrastructure to be realized, security risks, and others. Therefore, enterprise architects can add value by translating business strategy to wider-ranging architectural decisions. This ensures a coherent portfolio of initiatives is maintained and prevents projects or teams building their own little agile silo.

The Enterprise Architect's Role in Agile Development

An Agile environment can be a challenging experience for any EA professional. Nonetheless, as we have explained so far, there is a place for architecture alongside Agile. Here are some clear-cut points architects ought to focus on to deliver value to the organization:

- Translate the business strategy into useful high-level technical initiatives without going to a granular level, thus ensuring the direction of development is in line with the business goals while stepping clear of micromanagement.
- Implement adequate standards, offering standardized interfaces to agile teams and enabling them to easily combine and reuse building blocks.
- Support management by providing an all-inclusive vision of the business' solutions and development initiatives.
- Convey this vision to the agile teams to ensure alignment with the strategic direction of the enterprise.
- Help to define the strategy for building and maintaining the architectural runway.
- Seek to continuously elevate modeling, design, and coding practices.
- Establish best practices by facilitating the reuse of code, components, and patterns that have proven to be successful.
- Organize workshops to encourage innovation.
- Generate and analyze innovative ideas around what technologies should be employed by the enterprise.
- Seek constant feedback on ongoing company-wide initiatives and run cost-benefit analyses where relevant.

Part 3: Agile Modeling

Models are an important instrument for architects and developers. They can provide important business context, help you analyze the impact of a change, compare and evaluate alternative solutions, and much more. Therefore, we want to take a look at the use of architecture models in the context of agile methods.

ArchiMate Models and Agile Ways of Working

In addressing the value and use of relating architecture models in agile ways of working, we focus specifically on SAFe as a representative agile framework and the ArchiMate language for enterprise architecture as the modeling approach. Please refer to the previous simplified figure of SAFe (Figure 1). It depicts several concepts used in SAFe as well as other agile methods – obviously, this is not an exhaustive representation of the framework. One of the salient aspects of this layering is that the higher up you go, the more the focus moves to the *intent*, rather than the *construction* of the solution.

In Figure 6, we show how you can use ArchiMate elements to express these concepts. For instance, at the *Portfolio* level, the strategy and high-level motivation elements from ArchiMate 3 (Course of Action, Capability, Resource, Goal and Outcome) can be used quite fruitfully to express the business intent and direction. Capability-based planning offers a good starting point for investment discussions at this and the next levels.

SAFe	ArchiMate
Portfolio Level	
Strategic Theme	Stakeholders, Drivers, Goals, Course of Action
Epic (Business, Enabler)	Outcome
Large Solution Level	
Capabilities, Enablers	Capabilities, Resources
Solution	Business & Application Services, Products, Business Processes
Economic Framework	Principles, Constraints
Program Level	
Features	Requirements, Application & Technology Services
Architecture Runway	Constraints, Core concepts
Program Increment	Plateau
Team Level	
User & Architecture stories	Requirements & Constraints, Application & Technology Functions
Product Increment	Plateau & Application Components, Artifacts, Deliverables

Figure 6: Example mapping of ArchiMate elements to SAFe concepts

At the *Large Solution* level, the high-level solution architecture is expressed in terms of relevant Capabilities and Resources, Products, Business and Application Services to be developed and

Business Processes to be supported, and the economic framework provides guiding principles and constraints.

At the *Program* level, the direction is expressed in Requirements and desired Services, the Roadmap by a series of Plateaus, and the Architecture Runway is expressed in ArchiMate's core concepts and relevant constraints such as technology standards. Finally, at the *Team* level we see User and Architecture Stories being expressed with Requirements and Constraints and the functions needed, while the Sprint Result (product increment) typically consists of Application Components, software Artifacts and other Deliverables, which result in a Plateau (a stable state of the architecture).

Note that details of the solution are typically not described in ArchiMate models. It is, after all, a language intended for Enterprise Architecture. Detailed design is usually done better using UML, BPMN or other implementation-level modeling languages. It is, however, quite easy to link your ArchiMate models to this implementation level.

On the one hand, several concepts in your architectural runway could be mapped directly to corresponding implementation-level elements, such as Application Components, Interfaces and Services in UML, or Business Processes in BPMN. The architectural backbone provided by ArchiMate models is invaluable in tracking the various dependencies between different initiatives, programs, and results, which, as already mentioned, is a great challenge to scaling Agile to the enterprise level.

This traceability between the different levels means that you know where changing strategies and requirements may have an impact, you can ensure the coherence and stability of your IT landscape, and you are demonstrably in control. The latter is increasingly important in highly regulated business environments.

Intentional Architecture

To be clear, this is not a recommendation to create big ArchiMate models to capture the entire design before you even start building anything. Architects should aim to create and evolve just-in-time models that capture the information necessary to make the right decisions at the right moment and convey the *intent* of the architecture to the development teams.

ArchiMate concepts are a great way to express this intentional architecture. That is because ArchiMate is *not* developed for detailed solution design. Its concepts are aimed at expressing the essential structural and behavioral aspects of solutions, and the motivation behind these. Moreover, ArchiMate spans across business and IT, and thus provides agile teams with context and clarity beyond just software development concerns. For example, in which business processes is this going to be used? Who are the prospective users and other stakeholders? What is the customer journey we envisage? How does it relate to other products in our portfolio? How does it

contribute to key business goals? For any organization (but especially in an agile context), everyone should understand the purpose of their efforts.

Providing this kind of higher-level context and intent is essential for agile development. In traditional waterfall environments, you might have had business analysts, information analysts and other roles with the specific task to translate strategic intent into concrete requirements. In agile development, however, much of this responsibility rests upon the shoulders of agile teams and product owners. Without a clear understanding of the context, how can you expect them to take the right design decisions?

Architecture Models to Support Agile Teams

Dealing with large, complex systems is difficult in any methodology, and is perhaps the biggest challenge in adopting agile ways of working. Due to their very nature, these large systems are not very agile, because the large number of dependencies they exhibit slows down change.

Some agile methodologists would say you 'simply' have to break down those systems into chunks that are each manageable by a single team. Microservices are an example of such an approach, but as anyone who has seen them in action knows, the result is often that the complexity that used to be hidden within the single large system now shows up in the connections and communication patterns between these microservices.

As Conway's law[1] states, organizations design systems that mirror their communication structures. This is quite obvious, since if your system architecture cuts across these communication structures, the extra effort of communicating between different teams often leads them to redesign the system structure so that each team has their own area of responsibility in the system architecture (e.g. a microservice), or to restructure teams to align with the architecture components to reduce this communication overhead.

From an agility point of view, though, such component teams are not always optimal. If you have, say, a three-layer architecture with a Web front end, a business logic layer, and a database, you may be tempted to have three corresponding teams. Many features, however, will require all three teams to collaborate, since these features often need something in the interface, some business logic, and some storage. This will lead to teams having to wait on each other, and hence lowers your speed of change and agility.

Agile methods therefore tend to favor cross-component, cross-disciplinary feature teams, consisting of experts on each of the parts of the architecture that are potentially touched by a feature, and also including e.g. experts on UX, testing, and DevOps. Such feature teams can tackle a feature end-to-end. This comes at the expense of some efficiency. Moreover, if some specialized

[1] https://en.wikipedia.org/wiki/Conway%27s_law

expertise (say on security, database technology, etc.) is scarce and needed by multiple teams, you run into limitations.

Since both approaches have their pros and cons, most organizations tend to have a combination of feature and component teams based on the specifics of their solutions. Deciding what works best strongly depends on your architecture.

Modeling Features and Components in ArchiMate

With ArchiMate models, you can serve both approaches. In fact, the subdivision between the active structure and behavior aspects in ArchiMate is quite similar to that between components and features. In the end, you need both, but you can choose to start with one or the other in organizing the work. Below, you see the full ArchiMate framework, with one application-layer example that illustrates these aspects.

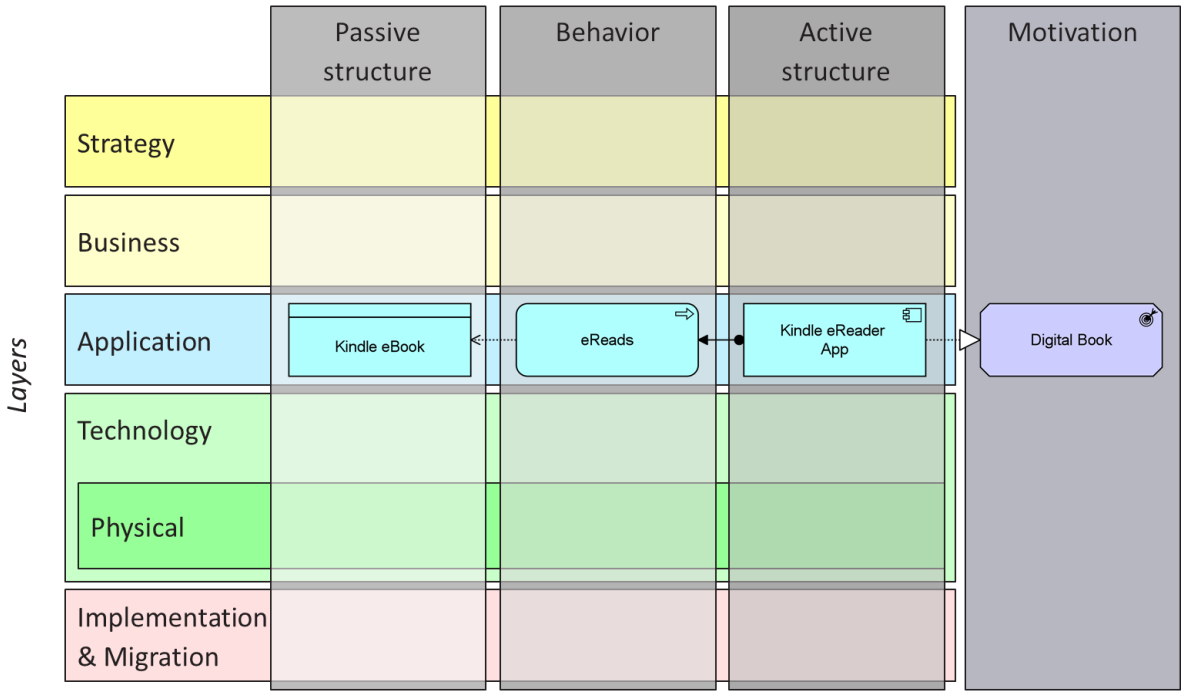

Figure 7. ArchiMate Framework

For a newly to be developed solution you typically start thinking from a feature perspective. In ArchiMate, you start with motivation concepts to capture the goals, requirements and planned outcomes for the enterprise and its stakeholders.

Next come ArchiMate's behavior concepts, with services, functions and processes. As the standard explains, an application service is explicitly defined as exposed application behavior that is meaningful from the point of view of the environment. In short, it represents what an application does for its users. This is the key concept for modeling software features in the agile sense. You shouldn't confuse this with the use of 'service' in a technical sense, e.g. as in a microservice

actually being a kind of software component. Rather, ArchiMate's service notion has a business-oriented background, of one party providing a service to another.

Application functions and processes are used next, to model the internal behavior needed to provide these services, i.e. what happens behind the scenes. Which application components (or business roles and actors in the case of a non-IT-based part of the solution) perform that behavior, and via which interfaces you can get these services, are the final steps in this design process.

When creating models of an existing solution, it is often easier to start with the components and their interfaces, since those are often more 'visible' and concrete than services and functions. The latter are somewhat more abstract and may require some deeper analysis of these components. Given that most solutions are 'brownfield' and live in a larger landscape of existing systems, in practice you will typically use a combination of behavior-to-structure and structure-to-behavior modeling.

Supporting Agile Team Collaboration with Architecture Models

Whichever way agile teams are structured, you always necessarily separate some concerns and combine some others. Knowing and understanding these dependencies is key in any multi-team setup. You want to avoid dividing things that are closely coupled, i.e. have a high degree of dependency, over different teams. That would result in an inordinate communication overhead between these teams. And knowing these dependencies is of course also the first step to reducing them.

However you structure your teams and architecture, in all cases there is a need for communication that increases with organization (and system) size. On the one hand, this is 'vertical', to convey the enterprise vision so the teams align in the same direction, and to provide feedback and contribute innovative ideas up the food chain from development to management. A clear line of sight helps ensure that goals are clear to everyone involved and priorities are aligned with these goals. Although organizations may use all kinds of tools to keep track of their development initiatives, architecture modeling is the most efficient way to illustrate the connections between strategy, capabilities and development portfolio items.

Vice versa, feedback on the feasibility and desirability of those goals comes from the 'boots on the ground' in the agile teams, who are in direct contact with users, know about the possibilities and limitations of technology, and provide improvements, innovations and refinements to adapt to local circumstances. Architecture models provide such an unambiguous connection between motivation and realization, and allow you to trace dependencies and effects of changes in priorities, planning, etc.

Equally important is the lateral communication between teams that work on parts of the solution that need to communicate, provide an integral user experience, share resources, and in general depend on each other. A common technique to draw these dependencies is using red woolen

strings on a Kanban or Scrum board, but that won't scale up across larger teams and teams-of-teams (spanning these threads across your work floor from board to board would be interesting…). Moreover, you cannot easily use this in any remote setting of course. Again, architecture models are of help here, clearly showing how different components are interconnected, how business processes rely on IT systems, which users are working with which applications, where the data is stored, etc.

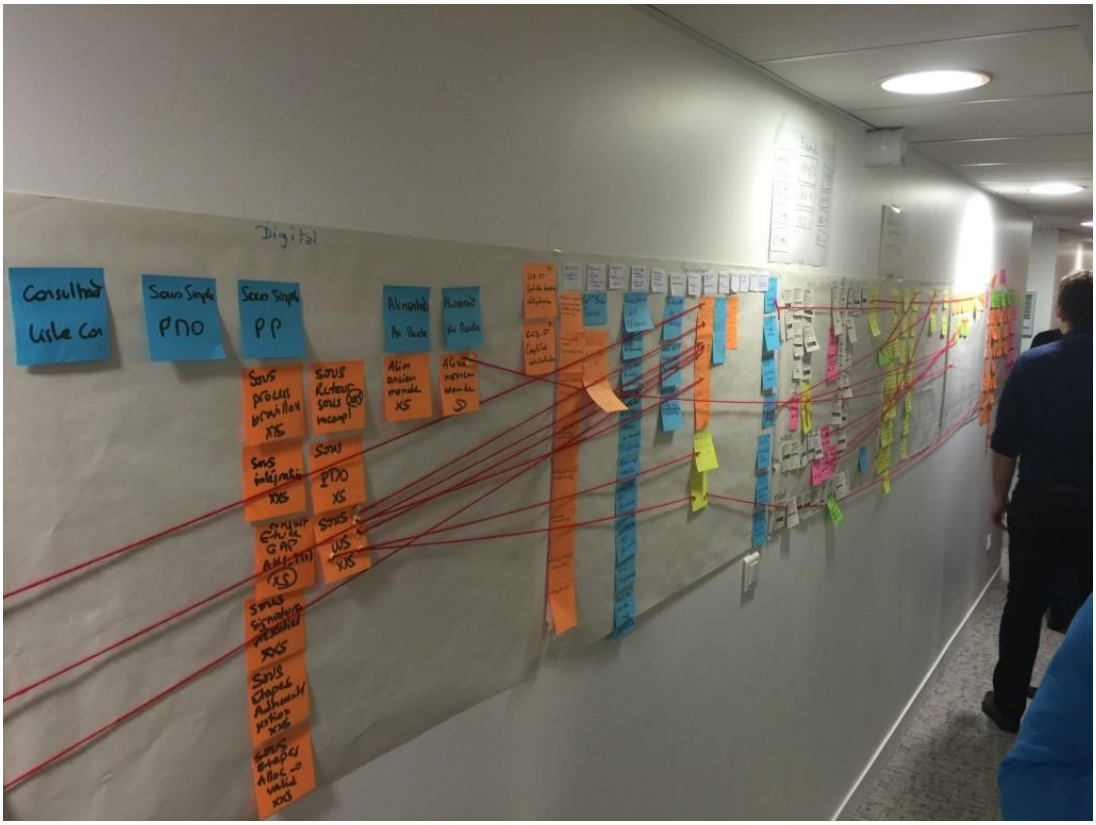

Figure 8. Dependencies on a Scrum board (source: blog.xebia.fr)

For both component and feature teams, knowing who is working on which parts of a solution is essential to avoid conflicts, gaps, overlaps, and rework. Typically, each team is fully responsible for the inner workings of what they build, i.e. its software architecture and design. They can use whatever they choose to design that, ranging from sketches on whiteboards to some simple ArchiMate models, and different teams may work differently. How these bits relate to each other is where things should be standardized in some way, so that all teams have the same understanding.

An overview of your architecture in terms of the key structure and behavior elements (i.e., components and features) and of their dependencies helps in defining team structure and supporting team communication. This is where, for example, a set of ArchiMate views of the communication between application components in your landscape can be a central design artifact. Moreover, managing and sharing these with an appropriate tool is a lot more convenient

than spanning strings across your workspace. An example of such an architecture landscape is shown below, anonymized and somewhat simplified from a true customer example in the insurance domain.

The figure highlights the impact of a new feature having to do with online insurance claim submission. This affects the website, the INSUR system and its subsystems, and a number of information flows. The image shows with colors which Agile/DevOps teams are responsible for which parts of this landscape, so you know who will have to collaborate on this specific feature.

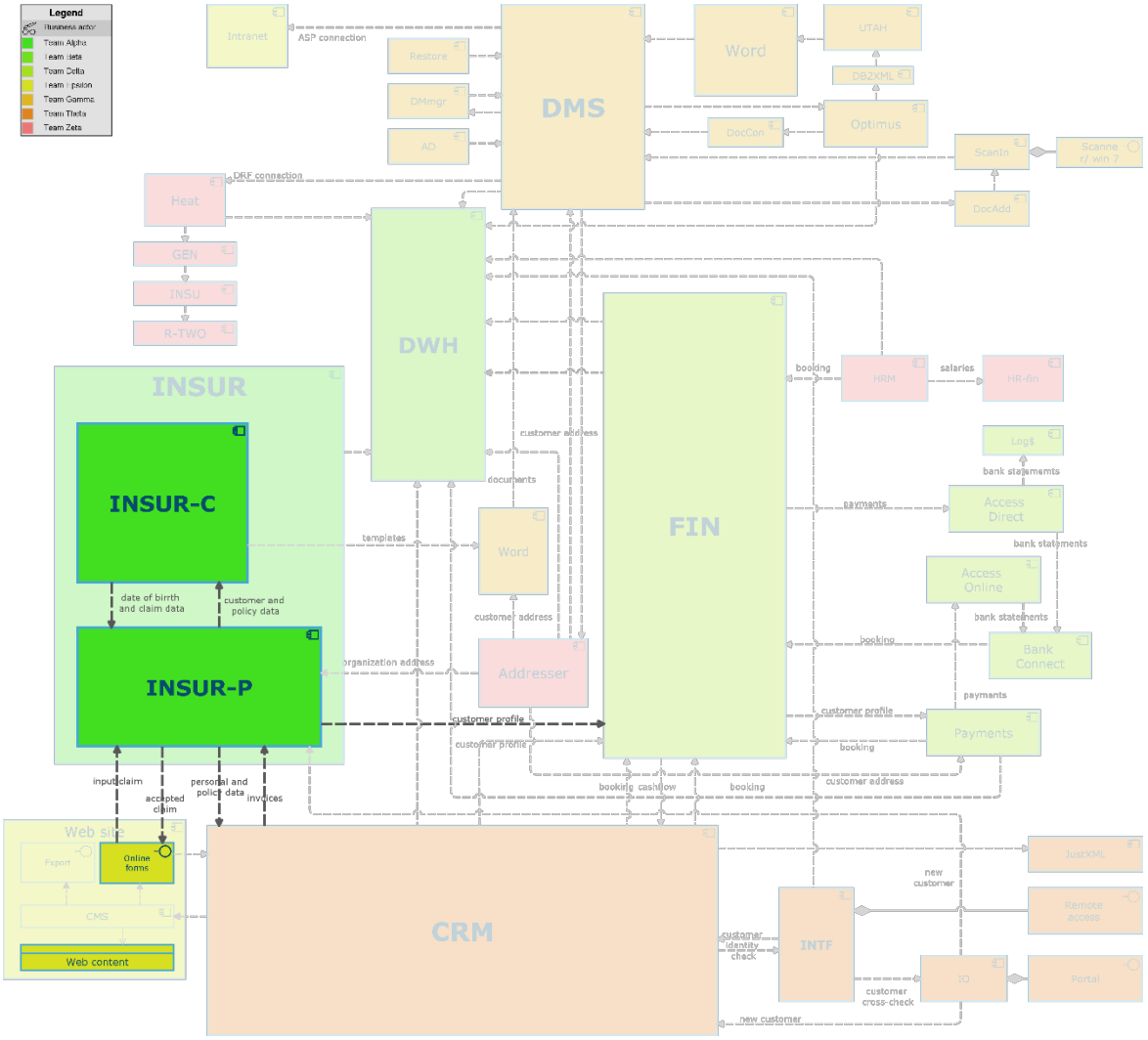

Figure 9. Application landscape, highlighting feature impact, coloring DevOps team responsibility

Now the example above takes a component view of the world and has teams for the main systems and their satellites, so such a view is perhaps be most relevant if you have component teams. In case of feature teams, focusing first on the application services that provide these features to users may make more sense. Drilling down from these application services leads you to the application functions necessary for providing them, and next you can identify which application components are involved in performing these functions.

Using an architecture model can also assist agile teams in several other ways, such as:

- Identifying effects of team-level planning decisions on the overall result, e.g. postponing some user story to a later sprint may affect another team's feature release date.
- Supporting decision making between teams from an end-to-end perspective, for example to ensure a unified user experience, avoid performance problems, address security and compliance concerns, etc.
- Sharing scarce resources such as specialized technical knowledge.

The Right Level of Detail

Probably the most commonly asked questions in modeling is: "What is the right level of detail for my models?" And in an agile context this is perhaps asked even more often. However, this is not something you can simply define up-front. There is no single, 'right' level of detail. This strongly depends on the context and objectives of your modeling effort.

Context is All

Of course, it is not a good idea to document everything in excruciating detail. Models need to be precise enough for their intended purpose. For example, to help team members understand a solution going forward; DevOps experts to deploy it and resolve issues (also think of integration testing, for instance); other teams to reuse components and services; and perhaps most importantly, anyone who needs to make changes in the future without having access to the original designers and builders. This is where the intent of the architecture is especially important.

Don't Model More Than You Can Maintain

Having models that are too detailed often leads to problems in maintaining them, simply because you cannot spare the effort needed to keep everything up-to-date. That in turn leads to models that age quickly, and sometimes people turning away from modeling altogether because the models they need to work with are always outdated anyway. This is clearly not the way to go.

So you should never model more than you are able to maintain (unless it is a one-shot, throwaway model). But how do you then decide on the level of detail?

Approach the 'Right' Level Iteratively

There are two complementary approaches to take. First of all, arriving at a good, useful level of detail is an iterative process in itself. You can use the agile process itself to improve this. In a retrospective you can discuss this and thus finetune your approach. Did you have the right information at the right time? Was the effort needed to create or update your models feasible and worth it? Like any other activity in an agile approach, modeling should be subject to this continuous improvement of your way of working.

Use a Risk-Based Approach

Secondly, you can use a risk-based approach to decide where you need more detail. Most agile methods are not very specific on this, but some of their predecessors like the Unified Process are

explicitly risk-focused and aim to address the most critical risks early on in the process. In line with this, you should create more detailed models for those areas of an architecture or design where the risks are greatest, and less detail where you can decide on the fly and the consequences of an error are limited. For instance, deciding on an infrastructural technology is often high-risk, because of the cost involved and to avoid the time and effort it will take to reverse that decision when it turns out to be the wrong choice after all. Other common high-risk areas are health & safety, security, and compliance. Think of personally identifiable information (PII) risk, financial transaction handling, manufacturing process control, etc.

No Single Level of Detail

This also implies that your models will not have a single, similar level of detail across the board. Moreover, even aiming for such a standardized level of detail may lead you to spend effort where it is not really needed, and conversely, to lack the time for those areas where it really counts. So any modeling approach that tries to tell modelers up-front what the 'right' level of detail is, will most likely result in models that are almost always at a 'wrong' level of detail – either too granular or too general.

Creating Architecture Models in an Agile Way

The previous sections discussed the use of architecture models in the context of agile development, and the 'right' level of detail for such models. But how do you go about creating models in an agile context?

First and foremost, the modeling process itself is something that should be done using the same agile principles that you use for developing software. That means that the creation of models is itself an iterative process that starts small and gradually extends models with content needed for specific purposes. This process should move in lock-step with the evolution of what these models describe.

Defining the Future

Some models are used as input for the design of some piece of software (or business process, product, etc.) and therefore need to be 'one step ahead'. As we know from agile ways of working, you want to postpone decision making to the last possible moment, when you have the best available information. That also holds true for models in an agile context. They are developed with a 'just in time' mindset, for example to provide context for developers in the next few sprints, or to help portfolio managers decide on priorities for the next epics, etc. This also means that the further ahead a model aims to look, the less concrete it will be. In ArchiMate terms, this translates into:

- models that only provide the longer-term, high-level intent in Motivation and Strategy elements;
- medium-term and medium-level models with more concrete details, for example to express main features as services, and to define concepts like an Architectural Runway;

- short-term, detailed models that are used, for example, to describe application components and their services and interfaces for developers who need to connect to these.

All of these models should be created by and with those who are directly involved, so the intent and ideas are captured at the source.

Understanding the Present

A second category of models are those that are created to comprehend the pre-existing, current situation. Most development is 'brownfield' and you need to understand this context. Models are helpful to get a handle on the complexity of such a current situation. For example, complicated communication patterns between microservices are very difficult to understand by just looking at the code. A model can clarify this.

Now, having to create those models of the current situation from scratch might be an inordinate effort. Ideally, those models would have been created and maintained by previous teams but that is often not the case. The second-best option is to automate some of that model creation. More and more tools exist to discover e.g. software components communicating on your network and that information can be used to build up a bottom-up view of the current state. Automated creation of, for example, ArchiMate models of the IT infrastructure based on information from those discovery tools, often linked to modern CMDB software, is how state-of-the-art teams work nowadays. At a business level, process mining tools can be used to discover the business processes of an organization. And we can expect more and more data analysis and AI-based techniques that help gain these kinds of insights and build up your models.

However, what you cannot reconstruct from just looking at the world out there is the intent behind what you see. What were the key design decisions and trade-offs? What alternatives were rejected and why? Who were the main stakeholders and what were their drivers? This is the main reason that you do need to document what you design and not just try to reverse-engineer that after the fact. This is again an argument for modeling intentional architecture in (ArchiMate) models.

Modeling What Was Created

That leads us to the third type of models, those that are intended to document what has been created after the fact so that others understand it. Those models are typically 'one step behind' in the development and realization process. Agile methods recommend to document late (but not too late!) so you capture what has actually been realized and you can incorporate relevant learning experiences, say feedback from customers, peer reviews or retrospectives. Moreover, the solution as realized may deviate from the solution as designed, and these differences are important to understand and discuss.

The Lifecycle of Models

All of these three kinds of models are useful, but they should be created and treated in different ways. The longer a model needs to live, the more thorough its development and maintenance should be. As I argued in a previous blog post, don't model more than you can reasonably maintain. Outdated information is often more dangerous than no information at all.

And there is a fourth kind of models not discussed here, of temporary models created during the creative design process. Those models often have a short lifespan and mainly serve to inspire, come up with new ideas, explain and let your peers validate your ideas.

Delivering Value Quickly

Below you can find some important pointers to keep in mind in order to minimize time-to-value and to deliver frequent, iterative return on investment through EA. We have illustrated them in ArchiMate to give you an actual demonstration of what we described in the previous section.

Prioritizing Work

To prioritize based on business value you can specify strategic themes in your architectural model as a starting point. The *Goal Contribution* view in Figure 7 shows two strategic themes of an insurance company, modeled as goals in ArchiMate (the top part of the figure). Different outcomes (at the bottom) contribute to these two strategic goals. Outcomes represent produced results and can, in turn, be linked to epics and enablers, which are pivotal elements in SAFe and agile development in general.

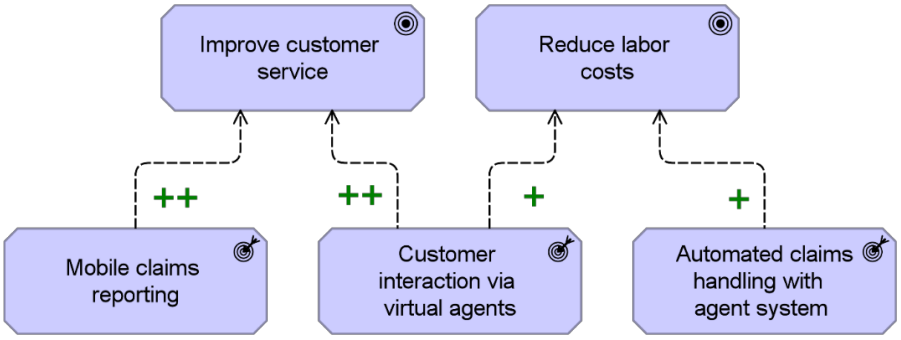

Figure 10: Goal Contribution view

In this model, two outcomes have a strong positive influence on the goal of improved customer service. They are an online claims form and a mobile application for communication with virtual insurance agents. The latter also contributes to the goal of reducing labor costs, an extra benefit. A new algorithm for automation of claims handling contributes to the same goal.

The clarity of the model allows architects to instantly derive a preliminary prioritization of outcomes from this representation. In this case, they learn that investing in the Virtual Agent capability will produce the highest business value for the insurance company, and as such should be prioritized over the other outcomes.

Subdividing Outcomes

To achieve top-down, enterprise-wide prioritization, you must take into account additional factors such as architectural dependencies within and across outcomes, or the assignment of epics across agile teams that are responsible for individual architectural elements. Enterprise Architecture proves very useful with this since architectural information provides a solid anchor for this type of big-picture planning.

As such, it is helpful to subdivide outcomes in an architecture-aware fashion. There is no need for a detailed specification of all requirements. The goal is to achieve just enough granularity to outline the organizational and architectural footprint of each outcome. This insight into the architectural footprint of the outcome is the key to finding conflicts and bottlenecks. Such information is usually not available in your typical agile management and tracking tools like Jira.

In the insurance example, the virtual mobile agent outcome can be split into two requirements that have a different architectural footprint. Both the mobile application component and the mobile back-end component are affected, and they are covered by different agile teams (see Figure 11 below).

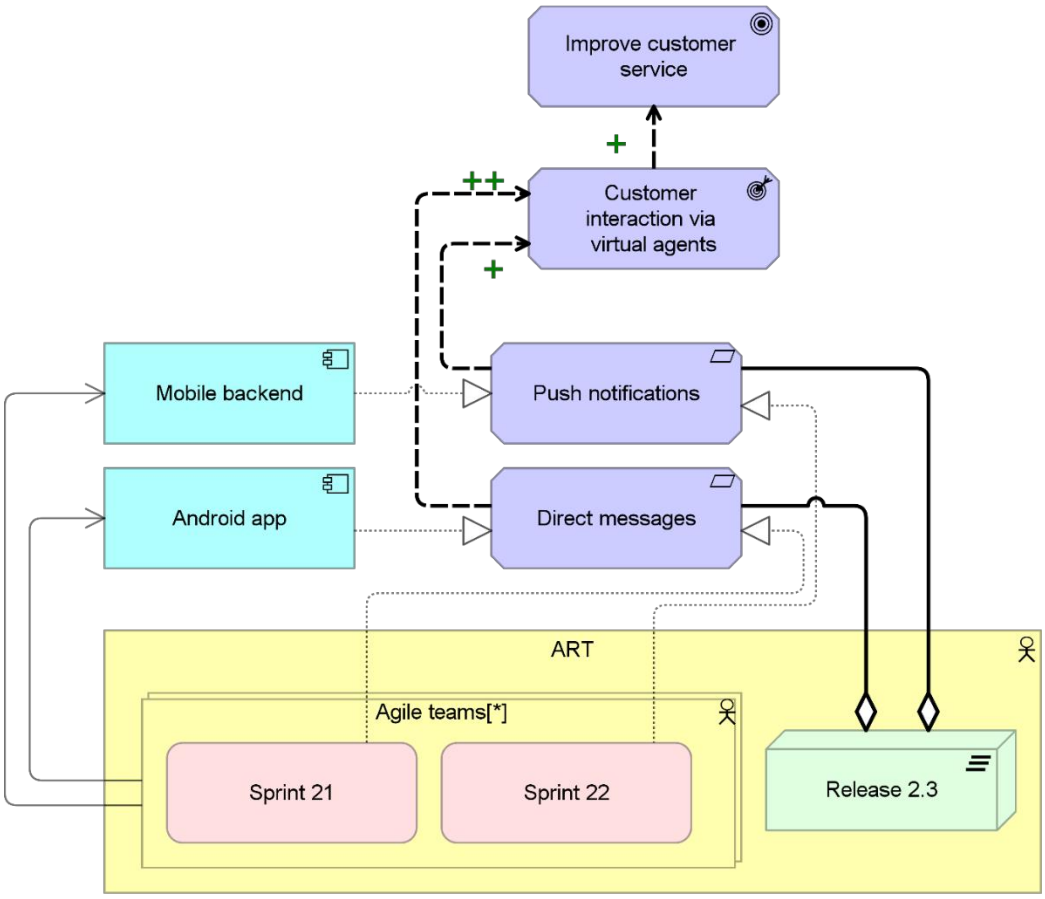

Figure 11: Development resources allocation

According to the sprint planning of the responsible agile team, the requirements are realized during different sprints. Based on the roadmap of the Agile Release Train (ART), both requirements will be part of the same release. The model clearly illustrates how business value is created over time and how development resources get allocated.

Combining EA and Agile in the HoriZZon Platform

At BiZZdesign we give a lot of thought to how we can lower barriers to workplace productivity to enable customers to create more value – this includes helping them integrate Agile in their EA practices successfully. There are multiple ways in which we support a continuous architecture mindset and practices. For an in-depth presentation of the HoriZZon platform please contact us, we will only briefly mention a few here.

We have built HoriZZon around the *immutable event stream* concept. This means all changes in the database are recorded as a stream of facts that do not change over time. The advantage of preserving this history is it allows users to create detailed views for specific audiences quickly and effectively. The views are derived from the data by processing all the changes that occurred up to a desired moment in the recorded event history.

By contrast, architecture frameworks have historically focused on sets of stakeholders and the viewpoints they need at different levels of detail, covering all of the architectural domains. Instead, our approach is compatible with the Agile mindset as it enables architects to engage in "just enough, just in time" architecture and offer stakeholders the exact views they need, in real time.

This marks a paradigm shift from a request-response mode of interaction to a subscribe-notify one, whereby individual stakeholders can receive personalized work news feeds, comment, provide feedback and in general collaborate in real-time around architecture and design information. Moreover, all kinds of other data streams can be integrated and correlated with your models, ranging from cost and performance to customer satisfaction and regulatory compliance.

This integration of external data sources also includes agile tools such as Jira and HoriZZon's API. This information can be aggregated at various levels and shared on the HoriZZon portal. The platform enables architects, developers and others to create various color views, roadmaps, and dashboards to inform stakeholders on the progress of changes and the performance of agile teams and release trains. Decisions and transition planning can be based on the available information on progress, prospective availability of new features, dependencies between these features, and associated business outcomes.

Users can calculate the effects of delays up-front and evaluate different scenarios when epics or features need to be reprioritized. This makes change more predictable and less risky as stakeholders focus on the true needs of the business.

Apart from these, the HoriZZon platform offers a great deal more features including support for best practice standards and frameworks, an intuitive modeling environment, deep analytics, as

well as customer journey mapping and asset lifecycle management capabilities. We will not get into all of these, but if you want to learn more about how to integrate Enterprise Architecture and Agile using the HoriZZon platform, do not hesitate to contact us.

Conclusions

The premise that architecture has a place alongside Agile is regarded by a good deal of industry professionals with skepticism. This stops many organizations from achieving their full potential. As Agile adoption grows, it becomes imperative that businesses realize the great complementarity between EA and Agile methods, and the valuable strategic contributions of which they are depriving themselves in the process by dismissing architecture. We believe an "either-or" vision only hurts competitiveness.

Instead, we are convinced that when adequately integrated the two practices can significantly increase business resilience and adaptivity. Part 1 of this guide argued in favor of their compatibility. Part 2 offered use cases and examples of best practices around integrating EA and Agile. Part 3 showed how architecture models can support agile development, both at the team level and to coordinate between different teams, scopes, levels of detail, and time scales. We hope that together this achieves this guide's goal of helping enterprise architects become more productive and deliver on the promise of Enterprise Architecture in Agile environments.