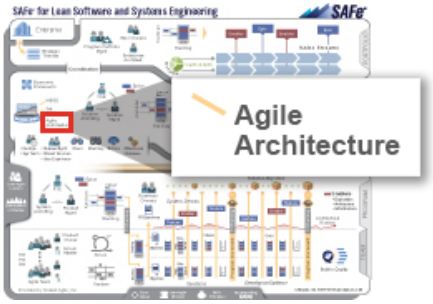
(https://twitter.com/ScaledAgile) (https://www.linkedin.com/grps/Scaled-Agile-Framework-4189072?)

HOME TRAINING COMMUNITY IMPLEMENTING CASE STUDIES RESOURCES INTRODUCTION TO SAFE

(https://www.youtube.com/user/scaledagile) (http://www.slideshare.net/ScaledAgile)

SAFe® PROVIDED BY SCALED AGILE

(http://www.scaledagileframework.com)



(/)

> *While we must acknowledge emergence in design and system development, a little planning can avoid much waste.*
>
> *—James O. Coplien, Lean Architecture*

Agile Architecture Abstract

Agile Architecture is a set of values and practices that support the active evolution of the design and architecture of a system, *concurrent with* the implementation of new business functionality. With this approach, the architecture of a system, even a large system, evolves over time while simultaneously supporting the needs of current users. This avoids the Big-Up-Front-Design and starting and stopping of stage-gated methods. The *system always runs*, and it thereby supports a more continuous flow of value.

Agile architecture enables incremental value delivery by balancing between two end points—emergent design and intentional architecture. *Emergent design* provides the technical basis for a fully evolutionary, incremental implementation approach, and it helps the design respond to immediate user needs. The design *emerges* as the system is built and deployed. At the same time, however, organizations must respond to new business challenges with larger-scale architectural initiatives that require some intentionality and planning. *Intentional architecture* provides guidance and technical governance to Agile programs and teams for certain overarching technical constructs. These provide for commonality of approach, elimination of redundancy, and higher robustness of the full system. Together, these activities create the *Architectural Runway* needed to enable teams to deliver business value faster

and more reliably.

SAFe's principles of agile architecture, described in this article, illustrate how collaboration, emergent design, intentional architecture, design simplicity, designing for testability, prototyping, domain modeling, and decentralized innovation all support this important underpinning to Lean-Agile development.

Details

By balancing emergent design and intentionality, *Agile Architecture* is a Lean-Agile approach to addressing the complexity of building Enterprise (/enterprise/) Solutions (/solution/). It avoids Big-Up-Front-Design, starting and stopping, and the big-branch-and-merge problems associated with traditional development. It's an approach that supports the needs of current users while simultaneously evolving the system to meet near-future needs. Used together, emergent design and intentionality continuously build and extend the Architectural Runway (/architectural-runway/) that provides the technical foundation for future development of business value.

Agile architecture spans all levels of SAFe. The following *Principles of Agile Architecture* provide the basic principles for the SAFe approach:

1. Design emerges. Architecture is a collaboration.
2. The bigger the system, the longer the runway.
3. Build the simplest architecture that can possibly work.
4. When in doubt, code or model it out.
5. They build it, they test it.
6. There is no monopoly on innovation.
7. Implement architectural flow.

#1 – Design Emerges. Architecture Is a Collaboration.

Traditional, stage-gated development methodologies often use Big-Up-Front-Design (BUFD) to create a roadmap and architectural infrastructure intended to fully address the needs of the future system. The belief is that a one-time effort could capture requirements and architectural plans sufficiently to support the system for years to come.

However, this approach comes with many challenges. One is the delay in starting implementation. A second arises when the planned architecture—a large set of speculative, forward-looking constructs—meets the real world. Soon enough, the designs become brittle and hard to change, and eventually a big-branch-and-merge to a new set of speculative assumptions is the routine course of action. SAFe addresses this by combining emergent design and intentional architecture, driven by collaboration.

Emergent Design

Principle 11 of the Agile Manifesto [2] is the primary driver behind the concept of emergent design: *The best architectures*

Principle 11 of the Agile Manifesto [2] is the primary driver behind the concept of emergent design: *The best architectures, requirements, and designs emerge from self-organizing teams.* This implies that:

- The design is grown incrementally by those who are closest to it.
- The design evolves hand-in-hand with business functionality. It is constantly tested and enabled by Refactoring (/refactoring/), Test-First (/test-first/), and Continuous Integration (/continuous-integration/).
- Teams rapidly evolve the design in accordance with the currently known requirements; the design is extended only as necessary to implement and validate the next increment of functionality.

This new practice of emergent design is effective at the Team Level (/team-level/). However, emergent design alone is insufficient when developing large systems.

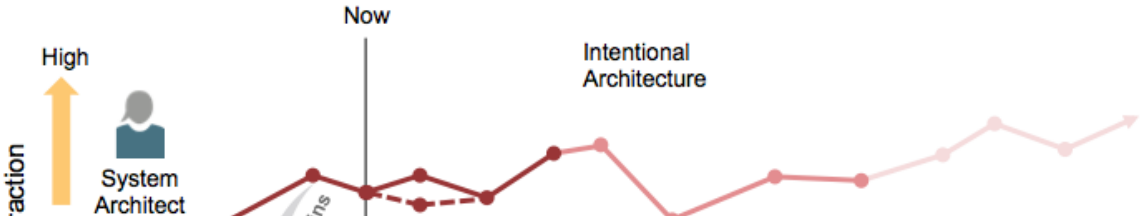
- It can cause excessive redesign for things that could have been anticipated. In turn, that drives bad economics and slows time to market.
- Teams are not always able to synchronize with each other, thus creating assumption entropy and architectural divergence.
- Teams may not even be aware of some of the larger, upcoming business needs; factors outside their purview drive the need for future architecture.
- Common architectural underpinnings enhance usability, extensibility, performance, and maintainability of the larger system of systems.
- New, cross-cutting user patterns affect future fitness of purpose.
- Mergers and acquisitions drive integrations and the need for commonality of infrastructure.

Intentional Architecture

Therefore, there comes a point at which emergent design is an insufficient response to the complexity of large-scale system development. Simply, it is not possible for teams to anticipate changes that may well occur outside their environment, nor for individual teams to fully understand the entire system and thereby avoid producing redundant and/or conflicting code and designs. For this some *intentional architecture* is needed—a set of purposeful, planned architectural initiatives to enhance solution design, performance, and usability and to provide guidance for inter-team design and implementation synchronization.

Architecture Is a Collaboration

Clearly, it's best to have both: fast, local control of emergent design and a global view of intentional architecture. The combination provides the guidance needed to ensure that the system as a whole has conceptual integrity and efficacy. Achieving the right balance of emergent design and intentional architecture drives effective evolution of the system, as Figure 1 illustrates.



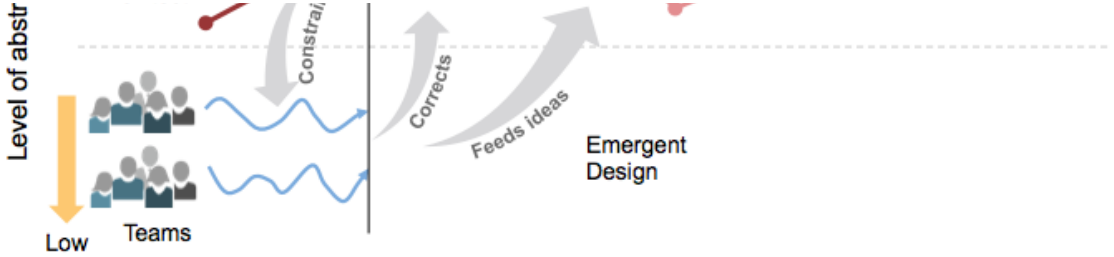


Figure 1. Intentional architecture, emergent design, and collaboration support system evolution

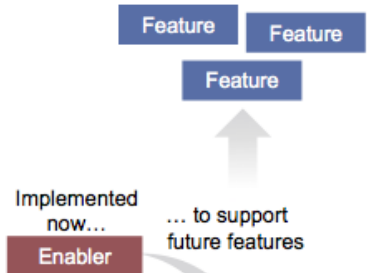
Figure 1 illustrates that these are not independent constructs. Intentional architecture constrains the emergent design, but at a high enough level of abstraction to allow the teams to effectively adapt the "intentional" part to their specific context. At the same time, emergent design influences and corrects intentional architecture and also feeds new ideas for future, centralized, intentional effort.

Such a deep reciprocity between emergent design and intentional architecture can occur only as a result of *collaboration* between Agile Teams (/agile-teams/), System and Solution Architecture (/system-and-solution-architect-engineering/), Enterprise Architects (/enterprise-architect/), and Product and Solution Management (/product-and-solution-management/). The Agile Release Train (/agile-release-train/) creates a team-of-teams environment that fosters this particular collaboration.

#2 – The Bigger the System, the Longer the Runway.

An architectural runway exists when the enterprise's platforms have sufficient technological infrastructure to support the implementation of the highest-priority Features and Capabilities (/features-and-capabilities/) in the backlog without excessive, delay-inducing redesign. In order to achieve some degree of runway, the enterprise must continually invest in extending existing platforms, as well as building and deploying the new platforms needed for evolving business requirements.

In the Lean-Agile enterprise, implementation of architectural initiatives is complicated by the fact that the "big-bang-up-front branch-and-merge" waterfall approach is abandoned. Instead, the enterprise commits to implementing architectural initiatives incrementally in the main code base. Doing so means that architectural initiatives must be split into Enabler Features (/enablers/), which build the runway needed to host the new business features, as Figure 2 illustrates.



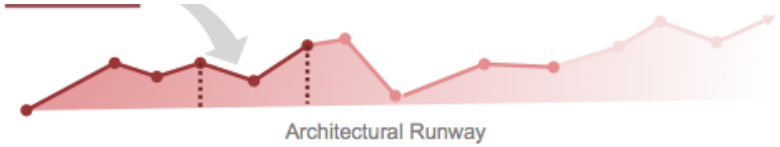


Figure 2. The architectural runway continuously evolves to support future functionality

Each enabler feature must be completed within a Program Increment (/program-increment/) such that the system always runs, at least at the PI boundaries. In some cases, this means that new architectural initiatives are implemented piecemeal and may not even be exposed to the users in the PI in which they are implemented. In this way, architectural runway can be implemented and tested behind the scenes, allowing shipment throughout, and then exposed to users when a sufficient feature capability exists in the next PI or so.

The concept of runway illustrates how intentional architecture and emergent design effectively complement each other at scale: intentional, high-level ideas in support of future functionality are adapted and instantiated by Agile Teams; they are empowered to figure out the optimal emergent design.

#3 – Build the Simplest Architecture That Can Possibly Work.

We welcome changing requirements even late in development (Agile Manifesto [2]). Yes, we do, but surely enabling change is facilitated by systems that have understandable designs. As Kent Beck notes, "If simplicity is good, we'll always leave the system with the simplest design that supports its current functionality" [3]. Indeed, at scale, design simplicity is not a luxury but a survival mechanism. There are many considerations that help accomplish this. Here are a few:

- Use a simple, common language to describe the system
- Keep the solution model as close to the problem domain as possible
- Refactor continuously
- Ensure that object/component interfaces clearly express their intent
- Follow good old design principles [4, 5]

Certain approaches to designing the system, such as Domain-Driven Design [6], usage of design patterns [4], and applying metaphor [3], simplify both the design and the communication between the teams. This "social" aspect of design simplicity is critical as it enables collective code ownership, which in turn fosters feature—rather than component—orientation [7]. The dominant theme in evolving maintainable and extensible solutions is to consider the system as a set of collaborating entities. This prevents typical design flaws such as concentrating too much logic at the database layer, or creating a thick UI, or ending up with large and unmanageable classes. Simplicity requires design skill and knowledge. Communities of Practice (/communities-of-practice/) help develop and spread these best practices.

#4 – When in Doubt, Code or Model It Out.

Coming to agreement on good design decisions can be difficult. There are legitimate differences of opinion about which course of action is best. Often there is no one right answer. And while Agile Teams and programs don't mind refactoring, they surely want to avoid *unnecessary* refactoring.

In order to decide, Agile teams can typically just *code it out*, using Technical or Functional Spikes (/spikes/) and even rapid prototyping. Iterations (/iterations/) are short and spikes are small; the result is fast feedback with objective evidence, which can then be subject to A/B testing by the teams, designers, and architects—or even the users.

In cases where design changes are of such scope that coding, spikes, and prototyping are insufficient, it is useful to *model* the problem to gain an understanding of the potential impact prior to implementation. As illustrated in Figure 3, Domain Modeling (/domain-modeling/) (the subject of a Guidance article) and use-case modeling are two lightweight Agile modeling constructs that are particularly valuable.

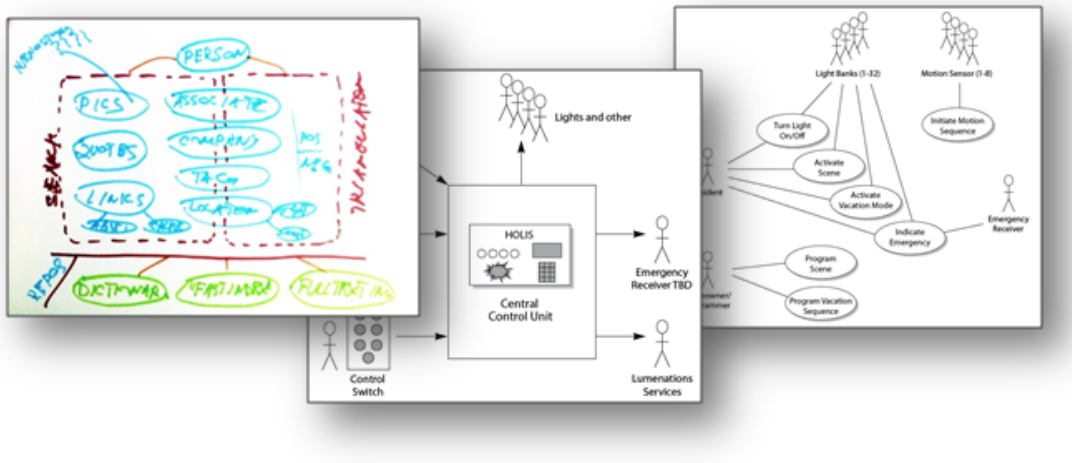


Figure 3. A domain model and a use-case model, supported by a system context diagram

Record Models in Solution Intent

Of course, the models aren't useful if no one can find them. Therefore, models, technical knowledge, and even various design decisions are recorded in the Solution Intent (http://dev.scaledagileframework.com/solution-intent/), which provides a central point of communication. In practice, however, system builders represent technical information in many forms, from documents to spreadsheets to the models described above. While documents and spreadsheets are easy for an author to create, they do not necessarily encourage the knowledge transfer or the continuous collaboration required for Lean systems engineering.

A better approach is Model-Based System Engineering (/model-based-systems-engineering/) (MBSE), which is the topic of another article.

With MBSE, solution intent will contain many different kinds of models, with many options for organizing and linking information. System and Solution Architects and Engineers are typically responsible for tasks ranging from specifying the model information and

organization to ensuring its quality. Agile Teams populate the model(s) with their respective knowledge and information.

#5 – They Build It, They Test It.

The responsibility for knowing whether a design actually works rests with those who collaborate on the architecture. Testing system architecture involves testing the system's ability to meet its larger-scale functional and Nonfunctional (/nonfunctional-requirements/) operational, performance, and reliability requirements. To do this, teams must also build the testing infrastructure—automate wherever possible—that enables ongoing system-level testing. And as the architecture evolves, the testing approaches, testing frameworks, and test suites must evolve with it. Therefore System Architects, Agile Teams, and the System Team (/system-team/) actively collaborate in order to continuously Design for Testability (/design-for-testability-a-vital-aspect-of-the-system-architect-role-in-safe/).

#6 – There Is No Monopoly on Innovation.

Optimization of architecture is a collaborative effort of Agile Teams, architects, engineers, and stakeholders. This can help foster a *culture of innovation* whereby innovation can come from anyone and anywhere.

Though such ideas come from anyone, capturing and propagating them requires some centralization via communication and recording in system intent. One of the responsibilities of the Enterprise Architect is to foster an environment where innovative ideas and technology improvements that emerge at the team level do not pass unnoticed but can be synthesized into the building blocks of the architectural runway. And as Agile can foster the "tyranny of the urgent iteration," programmatic time for innovation should also be built into occasional Innovation and Planning Iterations (/innovation-and-planning-iteration/).

#7 – Implement Architectural Flow.

Enterprise-scale architectural initiatives require coordination across Agile Release Trains and Value Streams (/value-streams/). Effective flow of these architectural initiatives is made visible via the Portfolio Kanban (/portfolio-kanban/). There, program and value stream enabler features and capabilities follow a common work flow pattern of exploration, refinement, analysis, prioritization, and implementation. In addition, the "pull" nature of the Kanban (/team-kanban/) system allows programs to establish capacity management based on WIP limits. This helps avoid overloading the system.

Together with the portfolio Kanban, the Program and Value Stream Kanban (/program-and-value-stream-kanbans/) systems provide a SAFe enterprise content governance model. This instantiates portions of the Economic Framework (/economic-framework/) and helps Decentralize Decision-Making (/decentralize-decision-making/), both of which are vital to a fast, sustainable flow of value.

Learn More

[1] Leffingwell, Dean. *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise.* Addison-Wesley, 2011.

[2] Manifesto for Agile Software Development. http://agilemanifesto.org/ (http://agilemanifesto.org/).

[3] Beck, Kent. *Extreme Programming Explained: Embrace Change.* Addison-Wesley, 2000.

[4] Bain, Scott. *Emergent Design: The Evolutionary Nature of Professional Software Development.* Addison-Wesley, 2008.

[5] Shalloway, Alan, et al. *Essential Skills for the Agile Developer: A Guide to Better Programming and Design*. Addison-Wesley, 2011.

[6] Evans, Eric. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, 2003.

[7] Larman, Craig and Bas Vodde. *Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum.* Addison-Wesley, 2010.

[8] Coplien, James and Gertrud Bjørnvig. *Lean Architecture for Agile Software Development.* Wiley and Sons, 2010.

Last update: 31 March 2016

The information on this page is © 2010-2016 Scaled Agile, Inc. and is protected by US and International copyright laws. Neither images nor text can be copied from this site without the express written permission of the copyright holder. Scaled Agile Framework and SAFe are registered trademarks of Scaled Agile, Inc. Please visit Permissions FAQs (http://scaledagile.com/permission-faq/) and contact us (http://www.scaledagile.com/permissions-form/) for permissions.

FRAMEWORK

Downloads (/posters/)

Latest Updates (/blog/)

SAFe 3.0 (http://v3.scaledagileframework.com)

TRAINING

Course Calendar (http://www.scaledagileacademy.com/events/event_list.asp)

About Certification (http://www.scaledagileacademy.com/?page=WhichCertification)

Become a Trainer (http://www.scaledagileacademy.com/?page=becomeatrainer)

PERMISSIONS

Permission FAQ (http://www.scaledagile.com/permission-faq/)

Permissions Form (http://www.scaledagile.com/permissions-form/)

Usage and Permissions (/usage-and-permissions/)

PARTNER

Becoming a Partner (http://www.scaledagile.com/become-a-partner/)

Partner Directory (http://www.scaledagile.com/listingcategory/directory/)

Partner Event Calendar (http://www.scaledagile.com/event-list/)

GET SOCIAL

Twitter (https://twitter.com/ScaledAgile)

Linkedin (https://www.linkedin.com/grps/Scaled-Agile-Framework-4189072?)

YouTube (https://www.youtube.com/user/scaledagile)

SlideShare (http://www.slideshare.net/ScaledAgile)

RECENT POSTS

New Essential SAFe guidance article (/new-essential-safe-guidance-article/)

Nov, 30th 2016

SAFe Website Updates and new Glosssary (/safe-website-updates-and-new-glosssary/)

Nov, 29th 2016

End of life for SAFe 3.0 website and courseware 12/30/16 (/eol-safe3/)

Nov, 28th 2016

SCALED AGILE, INC

CONTACT US