JIT-JEA

Capgemini Agile Architecture PoV

Part 3

Practicing JIT-JEA!

Capgemini

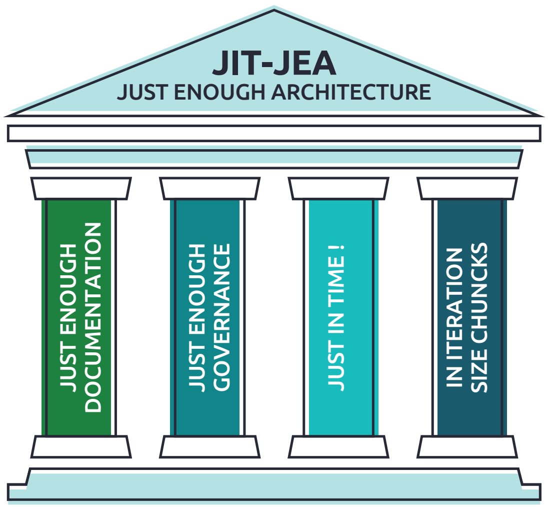
JIT-JEA Part 3: *Practicing JIT-JEA!*

TABLE OF CONTENTS

Welcome to the JIT-JEA Part 3: *Practicing JIT-JEA!*

After having published our first Agile Architecture Point Of View called JIT-JEA in January 2022 to introduce our five pillars model: "Just Enough Architecture", "Just Enough Documentation", "Just Enough Governance", "Just in Time" and "In iteration size chunks", we delivered the second one to talk about 10 real-life examples.

So in the first JIT-JEA, we talked about the "What", in the second Point Of View about the "How" and finally, in this third Agile Architecture Point Of View, "Practicing JIT-JEA !", we focus on the "With-What" and "When" aspects based on 10 Agile Architecture practices.



[illegible]

ALIASGAR MUCHHALA

Global Architects Community Lead Mumbai, IN

[illegible]

STEFANO ROSSINI

Italy Architects Community Chief Architect, IT

Introduction

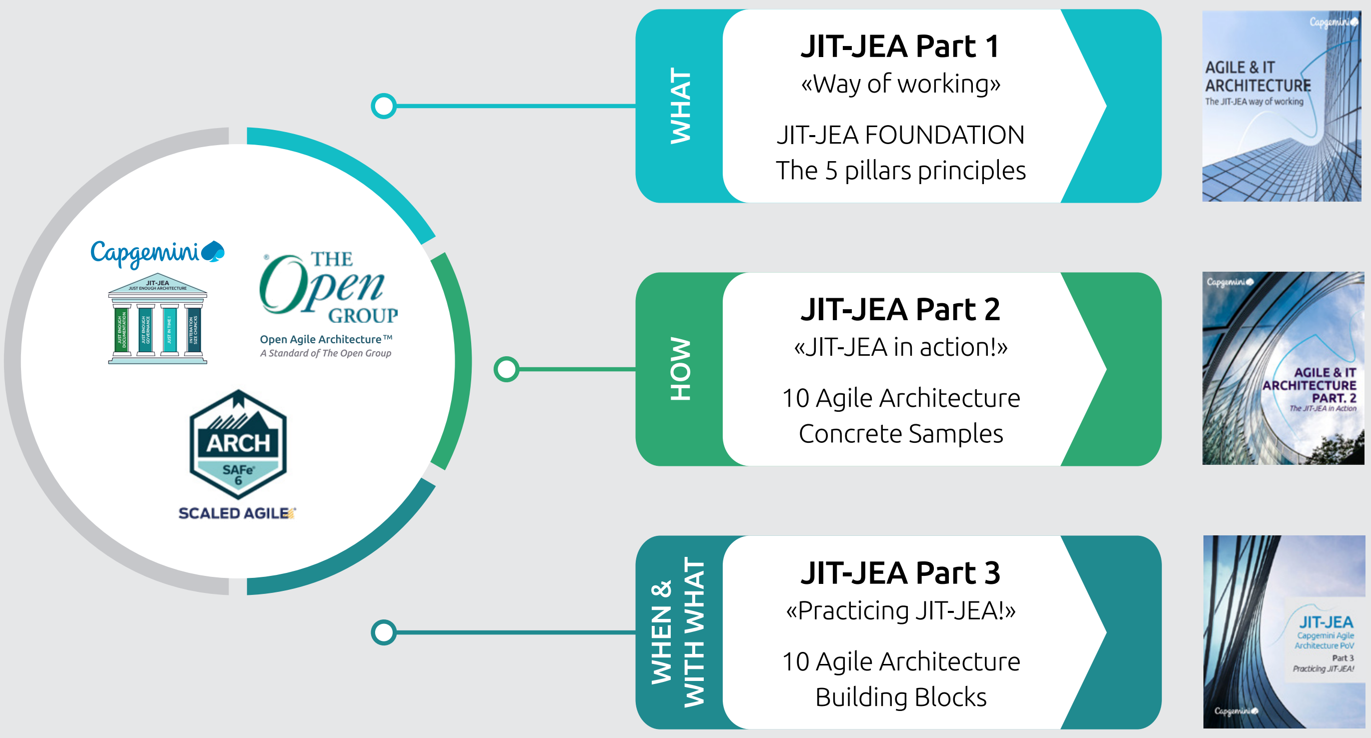
Agile architecture is the art of designing and delivering the "right" solution – meeting the requirements, expectations, and demands of the client – while being able to respond to a change in an uncertain environment, at an ever-increasing frequency.

In January 2022 we delivered the first Capgemini Point Of View about Agile Architecture called ***"The JIT-JEA way of working"***, introducing the concept of JIT-JEA: Just In Time, Just Enough Architecture (the *"what"*).

In the first Point of View, we defined the five pillars of Agile Architecture: Just Enough Architecture, Just Enough Governance, Just Enough Documentation, Just In Time, and In Iteration Size Chunk.

The second Capgemini Agile Architecture Point Of View called ***"JIT-JEA in Action!"*** was delivered at the beginning of 2023 and is a collection of 10 different real-life examples covering the five JIT-JEA pillars and principles, coming from the field of our architecture delivery projects (the *"how"*).

In this third Agile Architecture Point Of View, called ***"Practicing JIT-JEA!"***, we present 10 Agile Architecture Building Blocks in order to highlight when we should use them during a typical agile life cycle way of working (the *"when"*) and for each Architecture Building Block (the *"with what"*) we also make a cross-reference with the two important Agile Architecture references as Open Group Agile Architecture [O-AA] and SAFe Agile Architecture [SAFe-ARCH] and other useful references.



Bibliography

Capgemini Agile Architecture Point of View: The JIT-JEA way of working [JIT-JEA part 1]

https://www.capgemini.com/insights/research-library/agile-and-it-architecture/

Capgemini Agile Architecture Point of View: The JIT-JEA in action! [JIT-JEA part 2]

https://www.capgemini.com/insights/research-library/agile-and-it-architecture-part-2/

Open Agile Architecture [O-AA]

https://pubs.opengroup.org/architecture/o-aa-standard-single/

SAFe Agile Architecture [SAFe-ARCH]

https://scaledagileframework.com/agile-architecture/

Description of Architecture Building Blocks (With What - When)

Nr.	With What - Agile Architecture Building Block	When - Architecture Activity	Intentional Architecture	Evolving Architecture	Emerging Architecture
1	Architectural Guardrails & Fitness Functions	Manage technical complexity & risk (control & governance)	Defining guardrails	Monitoring & feedback loop Exception management	Request exceptions when required
2	Lightweight Architectural Decision Records (LADR)	Take architectural decisions within the given context	Decisions with high impact and high strategic importance	Decisions with high impact or high strategic importance	Decisions with low impact or limited strategic importance
3	Architect sync, GEMBA Walks, Management by Wandering Around (MBWA)	Collect and share feedback from delivery teams	Ensure emerging architecture aligns with the vision	Facilitate alignment between vision & actual delivery	Ensure the intentional architecture is realistic and feasible
4	Set Based Concurrent Engineering (SBCE)	Evaluate and develop solution options	Define evaluation framework	Analyse options with stakeholders	Propose alternative solutions
5	Flexible Release on Demand (RoD)	Decouple Deploy and release	Define the techniques for flexible RoD	Design the techniques for flexible RoD	Implement the techniques RoD
6	Architectural Patterns, e.g. Hexagonal, Clean Architecture, Event Driven Architecture (EDA)	Design for adaptability (flexible architecture)	Define adaptable architecture principles (e.g. loose coupling)	Design for adaptability & flexibility	Implement extensible design patterns
7	Architectural Enablers & Architectural Runway Continuous Architecture	Capture & define requirements (manage risk)	Establish requirement management	Quarterly plannings	Backlog grooming Sprint planning
8	Walking Skeleton, Minimum Viable Architecture (MVA), Sprint 0, Technology Radar	Research & learn (technology & trends awareness)	Define solution options	Explore solution options & collect feedback	Incorporate new and emerging technologies into the system's architecture
9	Principles & Data-Driven Insights	Developing an Architectural Roadmap Gather insights (understand & monitor the as-is)	Long-term roadmap with key architectural investments	Connecting individual short-term roadmaps with the long-term roadmap	Short-term roadmap for specific team/ product
10	Domain-Driven Design (DDD)	Gather insights (understand & monitor the as-is)	Alignment across value streams	Alignment at the program level	Value stream specific alignment

01

Manage Technical Complexity & Risk (Control & Governance)

Architectural Guardrails & Fitness Functions

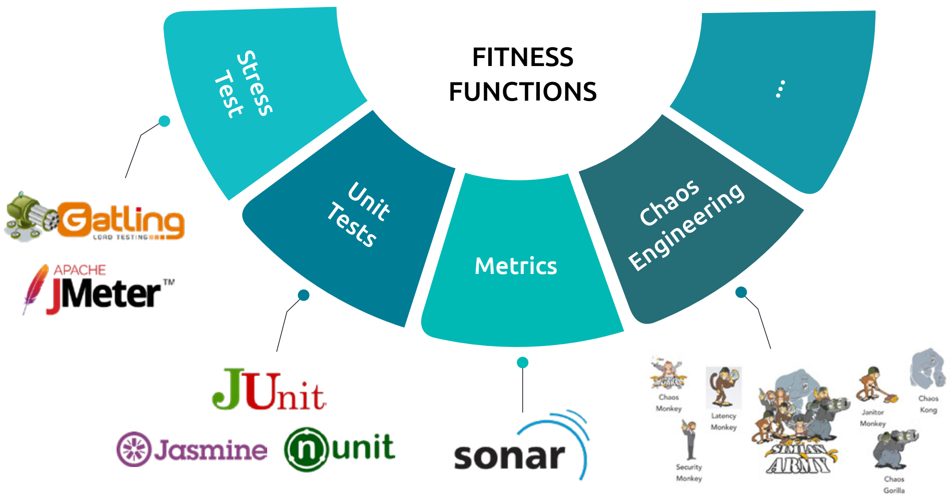
With What

Two mechanisms that organizations use to integrate evolvability into their system architectures are the concepts of Fitness Functions and Architectural guardrails.

Fitness Functions objectively assess whether the system is actually meeting its identified non-functional requirements (NFR); each fitness function tests a specific system characteristic.

Another important mechanism is the concept of architectural guardrails; as with their real-world roadside equivalents, software guardrails are designed to keep people from straying into undesirable territory.

In real terms, guardrails represent a lightweight governance structure. They document how an organization typically "does" things – and how, by implication, development teams are expected to "do" similar things. For example, a guardrail may document not just the specific availability requirements for a new service, but also how the organization goes about meeting such requirements; they can be patterns, good practices, or tools able to "guardrail" the implementation as SONAR, JUnit, JMeter, etc.



When

It is crucial to define architectural guardrails as part of Intentional Architecture, setting the foundation for guiding architectural decisions. The Evolving Architecture focuses on continuous monitoring through fitness functions to ensure alignment with these guardrails.

However, in the dynamic context of Emerging Architecture, exceptions may arise when strict adherence isn't feasible. Managing these exceptions and warnings becomes vital, requiring a well-defined process for requesting and reviewing them. This approach strikes a balance between control and flexibility, allowing Agile architecture to adapt to evolving needs while maintaining governance and mitigating risks effectively.

References

- **JIT-JEA Part 1: the 5 pillars** [JIT-JEA part 1]: 4.5 In Iterations size chunks
- **JIT-JEA Part 2: Agile Architecture in action!** [JIT-JEA part 2]: Sample #7 DevOps guardrails
- **Open Agile Architecture** [O-AA]:
  - Par. 6.3.1 Constraints/6.3.2 Fitness Functions/ 6.3.3 Guardrails
- **SAFe Agile Architecture** [SAFe-ARCH]: Lean Budget Guardrails: https://v5.scaledagileframework.com/guardrails/
- **Other references:**
  - Fitness functions or how to protect key characteristics from your product: https://continuous-architecture.org/docs/practices/fitness-functions.html
  - Building Evolutionary Architectures, N. Ford, R. Parsons, P. Kua, O'Reilly, 2017

02

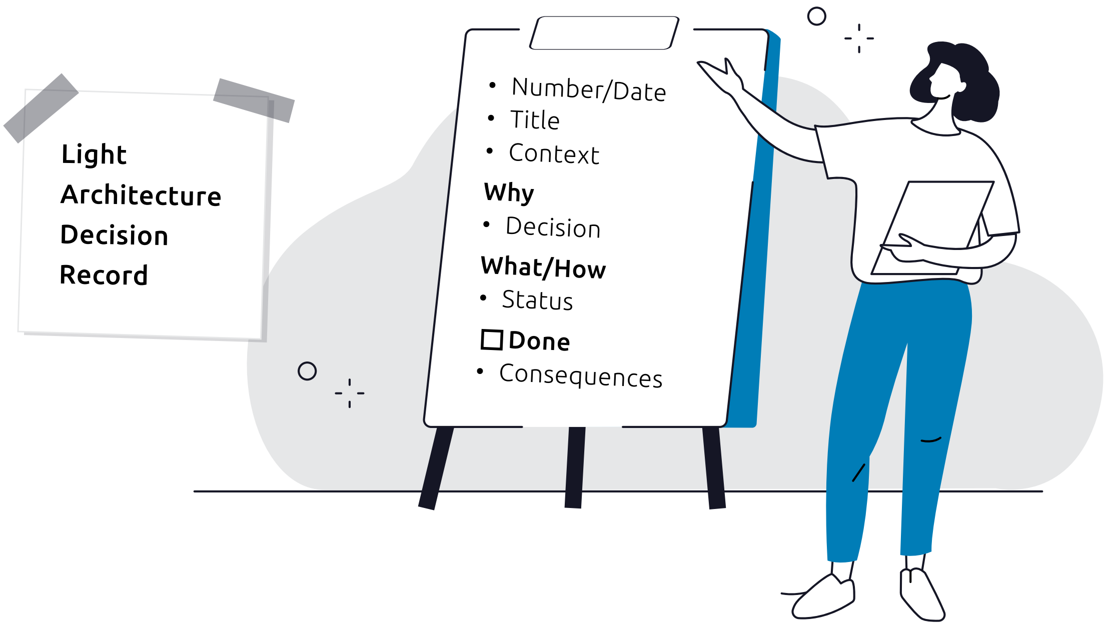
Take Architectural Decisions Within The Given Context

LADR - Lightweight Architectural Decision Records

With What

Agile methodologies do not discourage documentation but rather advocate for meaningful documentation. They emphasize the importance of avoiding excessive and unwieldy documents that often become outdated. Instead, Agile favors smaller, modular documents that are easier to maintain and to keep up to date.

In the realm of architecture, it is advisable to leverage Lightweight Architecture Decision Records (LADRs) to facilitate and document architecturally significant decisions. LADRs streamline the decision-making process, reducing the time and effort required. These concise records offer a more digestible format for all stakeholders while ensuring that they contain all the necessary information for decision-making, including capturing the architectural decision itself, the rationale behind and its associated impacts and consequences within a given context.



When

Documenting Architecture Decisions using Lightweight ADRs (Architectural Decision Records) is a valuable practice applicable across all architecture cycles: Intentional, Evolving, and Emerging.

In the Intentional Architecture cycle, decisions often span multiple streams and revolve around significant decisions. During Emerging Architecture, LADRs can be activated when encountering challenges or opportunities (Emerge) on which they need direction or guidance which is not in the Intentional or Evolving Architecture. Furthermore, Architecture Decisions may sometimes involve tactical solutions, leading to potential technical debt. LADRs serve a crucial role in documenting these decisions and ensuring they are addressed during subsequent refactoring efforts.

References

- **JIT-JEA Part 1: the 5 pillars** [JIT-JEA part 1] Par.4.3.5 Architecture Decision Record (ADR)
- **JIT-JEA Part 2: Agile Architecture in action!** [JIT-JEA part 2] Sample #3 LIGHTWEIGHT ARCHITECTURE DECISION RECORD (LADR)
- **Open Agile Architecture** [O-AA]:
  Par. 5.2. Architecturally Significant Decisions
  Par. 5.3. Architecture Decision Record
- **SAFe Agile Architecture** [SAFe-ARCH]: **NA**
- **Other references:**
  – Michael Nygard - *Documenting Architecture Decisions:* https://www.cognitect.com/blog/2011/11/15/documenting-architecture-decisions
  – Heiki W. Rupp (Red Hat) - Why you should be using architecture decision records to document your project: https://www.redhat.com/architect/architecture-decision-records
  – Architecture Decision Record: https://continuous-architecture.org/docs/practices/architecture-decision-records.html

03

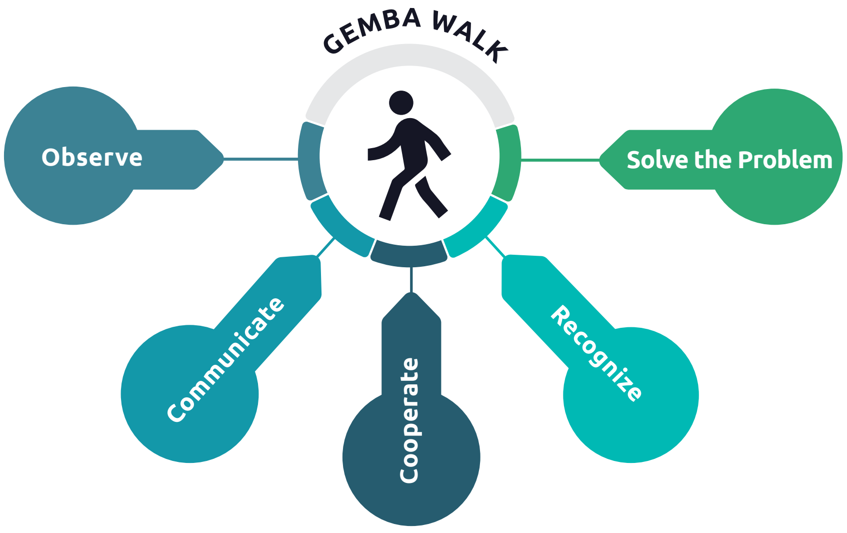
Collect and Share Feedback from Delivery Teams

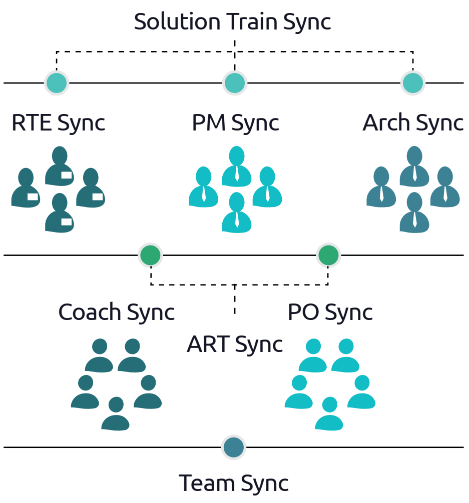
Architect Sync & GEMBA Walks

With What

“Going to the Gemba” is a powerful Lean management practice that plays a vital role in shaping the strategy formulation process. It means visiting the place (Gemba is a Japanese term meaning "the actual place") where value is created; for example, when clients engage with the enterprise or when employees deliver products and services. While data offers a representation of reality, it requires supplementation through real-world experiences. In essence, a map, or data, is not a perfect reflection of the actual landscape, or reality.

Agile architects play a crucial role in maintaining the balance between Intentional and Emergent Design throughout each iteration. They achieve this by evaluating the outcomes of enabler work, which encompasses new knowledge acquisition and additions to the architectural runway. Architects stay aligned and share progress and concerns at the Architect Sync event.





When

Conducting “Gemba walks” should become a seamless, ingrained process. Utilizing the insights and feedback garnered from these walks is the best practice across all architecture stages.

In Intentional Architecture, these walks serve to refine the architectural vision and enhance guidance towards the teams. In Evolving Architecture, they aid in aligning the intended architecture with the emerging one. In Emerging Architecture, Gemba walks provide valuable insights into the feasibility and realism of the intended solution for the teams.

To maintain a balanced approach, it's essential to integrate Gemba walks into the daily routines of both the Intentional and Evolving / Emerging Architectures.

References

- **JIT-JEA Part 1** [JIT-JEA part 1]: **the 5 pillars:** 4.3.4 Architecture Community of Practice [JIT-JEA part 1]
- **Open Agile Architecture** [O-AA]: Par. 11.1.1. Tenet 1. Situational Awareness
- **Agile Architecture** [SAFe-ARCH]: https://scaledagileframework.com/agile-architecture/
- **Continuous Exploration:** https://scaledagileframework.com/continuous-exploration/
- **Other references:** Gemba: https://en.wikipedia.org/wiki/Gemba

04

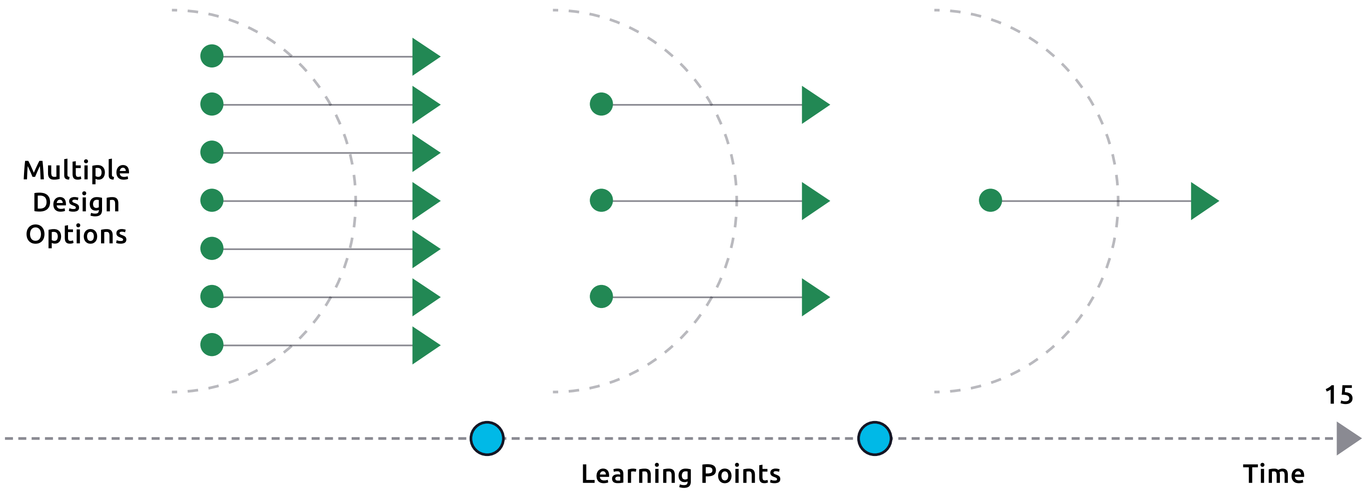
Evaluate and Develop Solution Options - SBCE-Set

Based Concurrent Engineering

With What

SBCE-Set-Based Concurrent Engineering is an approach to evaluate multiple product architecture alternatives and to delay architecture decisions until *"the last responsible moment"*.

Architects and developers initially cast a wider design net, considering multiple design choices at the start. After that, they continuously evaluate economic and technical trade-offs, typically exhibited by the objective evidence presented at integration-based learning points. Then they eliminate the weaker options over time and ultimately converge on a final design based on the knowledge gained till that point.



When

SBCE is a valuable approach when important architecturally significant decisions must be made. These decisions are typically made by one or several teams and need to be coordinated across teams. Intentional architecture supports this process as it's a purposeful set of statements, models, and decisions that represent some future architectural state. Therefore, SBCE should be started as part of the Intentional Architecture.

In the overall architecture process, the emerging architecture may require decisions coming from intentional architecture to change. New alternatives can be added, and past decisions can be reversed. Therefore, it is important to document the motivations behind past decisions, e.g. in a Lightweight Architecture Decision Record (LADR).

To make maximum use of the time to arrive at a maximum of progressive insights, SBCE will have to be started as soon as possible and individual decisions will have to be postponed until the Last Responsible Moment.

References

- **JIT-JEA Part 2: Agile Architecture in action!** [JIT-JEA part 1]: Sample #8
- **Open Agile Architecture** [O-AA]: Par. 5.7. Set-Based Concurrent Engineering (SBCE)
- **SAFe Agile Architecture** [SAFe-ARCH]: Principle #3 – Assume variability; preserve options: https://scaledagileframework.com/assume-variability-preserve-options/
- **Other references:** Allen C. Ward and Durward K. Sobek II: Lean Product and Process Development, Second Edition 2014

05

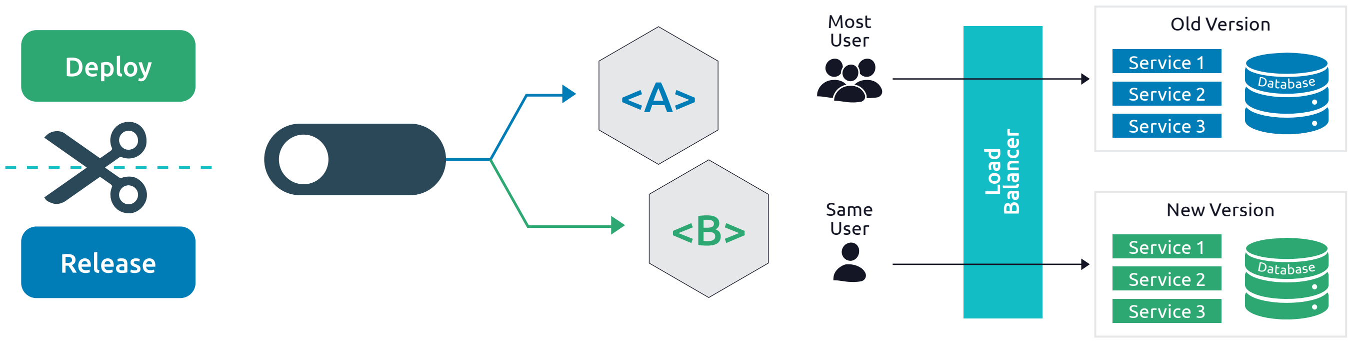
Decouple Deploy and Release

Flexible Release on Demand (RoD)

With What

In Agile Architecture it's important that the finalization of a release can be decoupled from the actual deployment. This enables addressing user-specific, on-demand activities at the most opportune times, aligning with when users require them or when it offers the greatest economic benefit to both customers and the business. For these different reasons to have Flexible release on Demand several techniques are available:

- Feature toggles – Provides a mechanism that allows code to be turned "on" or "off" without needing additional deployment, facilitating for example A/B testing of functionality.
- Dark launches – Like Feature toggles, a technique to selectively release/deploy new features into a production environment without releasing the functionality to all end users.
- Canary releases – The practice of releasing the solution to a specific customer segment and measuring the results before expanding and releasing it to more customers.



When

As mentioned, several techniques are available for creating Flexible Release on Demand. The selection of technique depends on the specific required flexibility and with that, the decision/ guideline to create such flexibility. Typically, Dark Launches and Canary Releases are already decided during Evolving Architecture and maybe even in the Intentional Architecture. Feature Toggles that decouple Deploy and Release are part of the Emerging Architecture with the required Cross stream alignment in the Evolving Architecture.

References

- **JIT-JEA Part 2: Agile Architecture in action!** [JIT-JEA part 2]: Sample #6
- **Open Agile Architecture** [O-AA]: 6.4.1.1. Feature Toggles
- **SAFe Agile Architecture** [SAFe-ARCH] Release on demand: https://www.scaledagileframework.com/release-on-demand/
- **Other references:**
  - Feature Toggles (aka Feature Flags): https://martinfowler.com/articles/feature-toggles.html
  - Canary: https://en.wikipedia.org/wiki/Canary

06

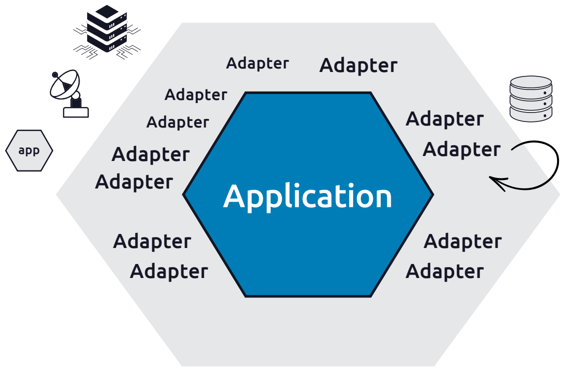
Design for Adaptability (Flexible Architecture)

Architectural Patterns

With What

In an Agile Architecture, it's crucial to create a flexible design that can adapt easily and swiftly to changes. Architectural designs such as Hexagonal architecture, Onion architecture, and Event-driven Architecture, which are grounded in the clean code principles advocated by Robert C. Martin ("Uncle Bob"), aid in achieving this objective.

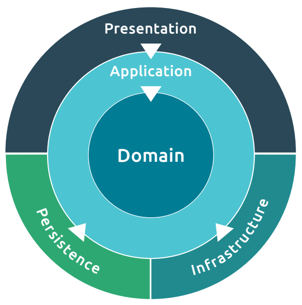
Hexagonal architecture: Created by Alistair Cockburn, it brings an interchangeability of adapter implementation and, therefore, a great suppleness in composing the domain with various ways of interacting with the software, as well as implementing infrastructure capabilities. The Hexagonal architecture allows an effective decoupling of application, domain, and infrastructure concerns.

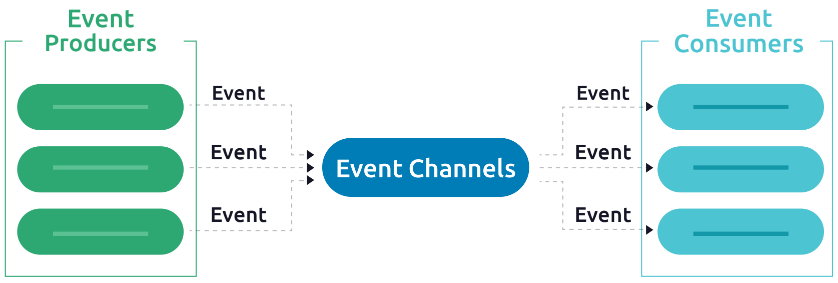


The Onion architecture: This has a fundamental rule that all code can depend on layers more central, but code cannot depend on layers further out from the core. In other words, all coupling is toward the center.

Event-Driven Architecture (EDA) is an architecture centered around the concept of an "event".

This approach leverages the immutable nature and decoupling capability of events, serving as a robust foundation for designing and developing domain logic. Its emphasis on loose coupling and strong autonomy, which prioritizes behavior and adaptability, underscores its value as a best practice within agile architecture frameworks.





When

The intentional architecture outlines principles and patterns that are crucial for ensuring architectural adaptability within the environment. To maintain alignment with these principles, guardrails and fitness functions are set up, ensuring that Evolving Architectures preserve their flexibility (e.g., as an integral part of CI/CD pipelines). Teams involved in developing the Emerging architecture need comprehensive training in these patterns to ensure accurate implementation that aligns with the intentional architecture.

References

- **JIT-JEA Part 2: Agile Architecture in action!** [JIT-JEA part 2]: Sample #4: Architecture Styles enabling design flexibility
- **Open Agile Architecture** [O-AA]:
  - Par. 21.3. Hexagonal Architecture: Why? Benefits?
  - Par. 21.2. Event-Driven Architecture
- **Agile Architecture** [SAFe-ARCH]: https://scaledagileframework.com/agile-architecture/
- **Other references:**
  - **Hexagonal architecture:** https://alistair.cockburn.us/hexagonalarchitecture/
  - **The Onion Architecture:** https://jeffreypalermo.com/2008/07/the-onion-architecture-part-1/
  - **The Clean Architecture (from: The Clean Code Blog):** https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

07

Capture & Define Requirements (Manage Risk)

Continuous Architecture, Architectural Runway and Architectural Enablers

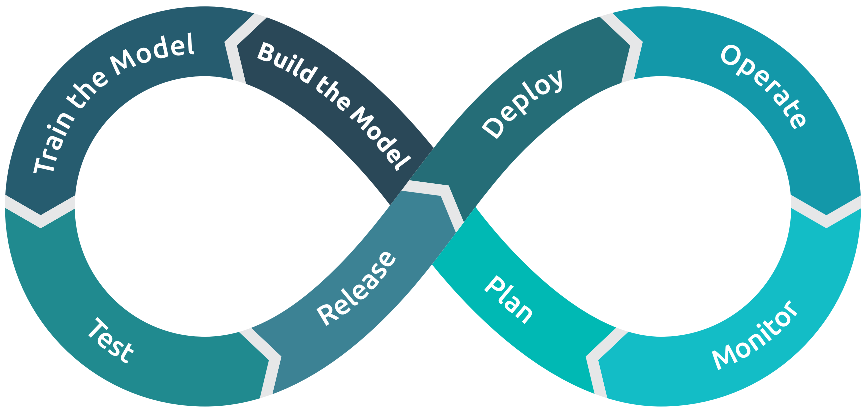
With What

Within an agile context, the architecture discipline evolves. It shifts from a Big Up-Front Design (BUFD) to **Continuous Architecture**. One of the things Continuous Architecture implies is that a small rapid change can be applied, which is important to better manage the uncertainty and complexity that characterizes the digital and agile transformation.

To minimize the risk of future functionalities failing to meet non-functional requirements and to ensure cross-cutting concerns are considered, it's essential to factor in future developments when defining the architecture. However, the question arises, how to capture and define requirements, manage priorities and risks in an appropriate way?

The **Architectural runway** designs the technical foundations needed to implement near-term features without excessive redesign; thus enabling a continuous flow of value.

Architectural enablers play a key role in constructing, expanding, and upkeeping the Architectural Runway. They are mainly utilized for exploration purposes, such as implementing architecture, refactoring, developing infrastructure, and ensuring compliance. These activities are made visible in the backlog for negotiation with the Product Owner.



When

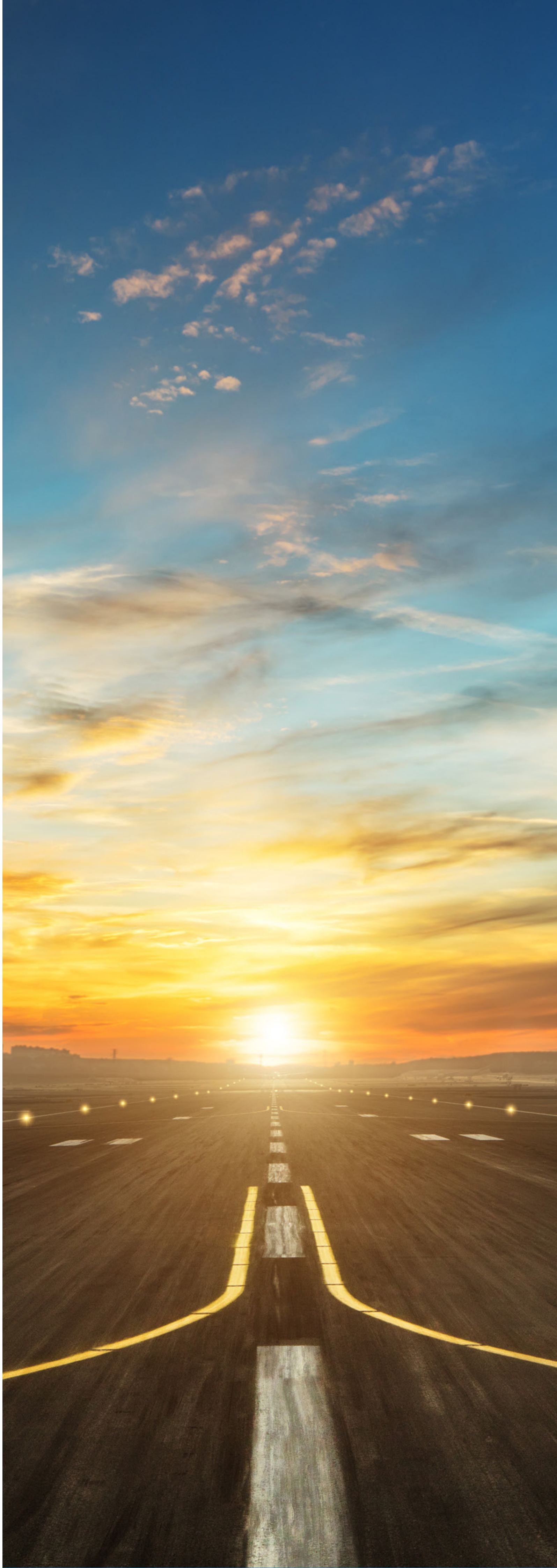
The intentional architecture establishes requirements management (including architecture enablers) by outlining the Architectural Runway, thereby supporting both current and future needs.

The evolving architecture evaluates the Architecture Runway as part of the quarterly planning by combining intentional and emerging architecture for the next program increment.

In the Emerging architecture, the content of the Architecture Runway is used during backlog grooming and for the actual sprint planning.

References

- **JIT-JEA Part 1: the 5 pillars** [JIT-JEA part 1]: par. 4.1.2 Architectural Runway / Enablers
- **JIT-JEA Part 2: Agile Architecture in action!** [JIT-JEA part 2]: Sample #1: Architecture Enablers / Sample #6: Architectural Runway
- **Open Agile Architecture** [O-AA]:
  - Par. 4.5.2. Intentional Architecture
  - Par. 4.5.3. Concurrent, Continuous, and Refactored
  - Par. 9.14. Axiom 14. Bias for Change
- **SAFe Agile Architecture** [SAFe-ARCH]:
  - **Enablers:** https://scaledagileframework.com/enablers
  - **Architectural Runway:** https://scaledagileframework.com/architectural-runway/
  - **Continuous Exploration:** https://scaledagileframework.com/continuous-exploration/
- **Architecting your product is a journey:** https://continuous-architecture.org/docs/practices/architecture-runway.html



08

Research & Learn (Technology & Trends Awareness)

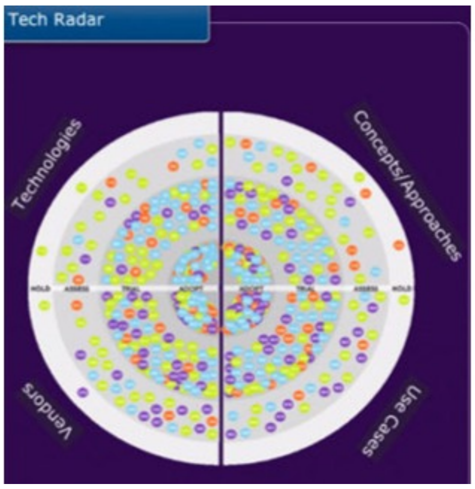
Technology Radar, Walking Skeleton

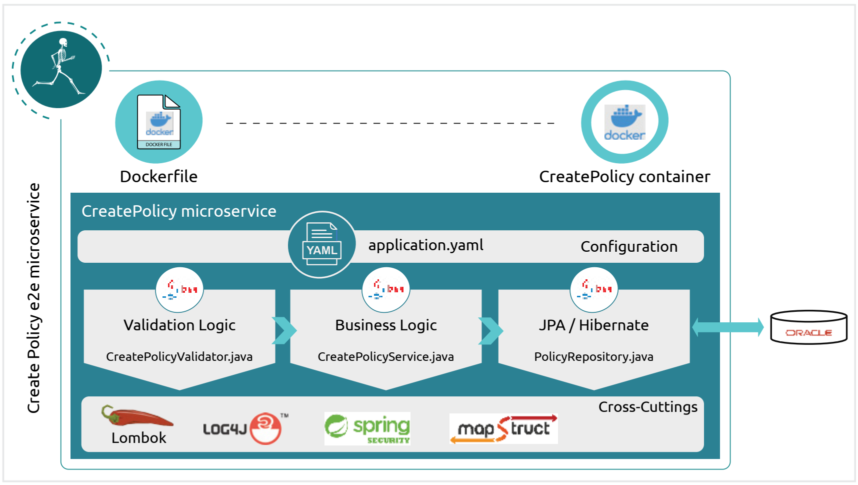
With What

In a world where technological change is accelerating and the business landscape is constantly evolving, Agile Architects must continuously research and learn about new technologies and trends to ensure systems meet the evolving needs of their organizations.

Walking skeletons are bare-bone representations of a system's architecture that embody the essence of agile development. When combined with proofs of concept (POCs/POVs) and prototypes, walking skeletons help architects build Minimum Viable Architectures (**MVAs**) by swiftly and efficiently exploring ideas, validating assumptions, and guiding initial design decisions, ensuring a robust architecture from the foundation.

Company Technology Radars can be used to identify new technologies and trends that may be relevant to the *walking skeleton* or *MVA*. This can help architects make informed decisions about the technologies to use and design the system to be flexible and adaptable to change.





When

In the Intentional Architecture, agile architects research and learn about the latest technologies and trends and use this knowledge to define the requirements and high-level solution options.

In the evolving architecture, agile architects build walking skeletons and MVAs, and use POCs/POVs and prototypes to validate design ideas and to get feedback on those. This feedback is used to refine the system's architecture and to ensure that it meets the needs of its users.

As part of the emerging architecture, agile architects incorporate new and emerging technologies into the system's architecture. Sprint 0, often overlooked, is a critical phase in a project's lifecycle. It's the preparatory phase where architects define the project's technical vision, ensuring alignment with organizational goals.

References

- **JIT-JEA Part 1: the 5 pillars** [JIT-JEA part 1]: par. 4.3.4 Architecture Community of Practice
- **JIT-JEA Part 2: Agile Architecture in action!** [JIT-JEA part 2] SAMPLE 10: Technology Radar for your company or division
- **Open Agile Architecture** [O-AA]:
  - Par. 5.5. From Intentional to Continuous
  - Par. 14.1. Defining Product Architecture
- **Continuous Exploration:** https://scaledagileframework.com/continuous-exploration/
- **Other references:**
  - Our Technology Radar: lifting the lid on how we build, ship and scale digital products: https://www.frog.co/designmind/technology-radar-how-we-build-ship-scale-digital-products
  - Walking skeleton (archived): https://web.archive.org/web/20140329201356/http://alistair.cockburn.us/Walking+skeleton
  - A Minimum Viable Product Needs a Minimum Viable Architecture: https://www.infoq.com/articles/minimum-viable-architecture/

09

Developing an Architectural Roadmap

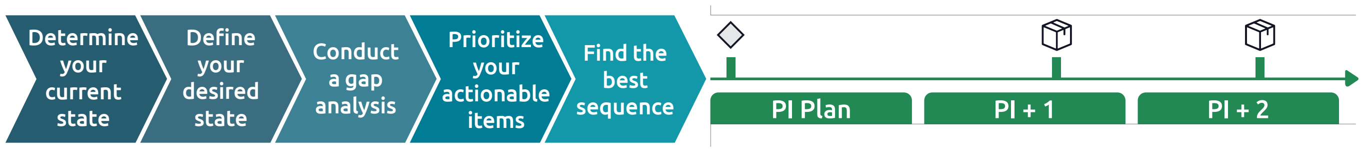
Principles & Data-Driven Insights

With What

Agile architects play a critical role in ensuring that company's technology investments are aligned with its business strategy and goals. To do this, they need to develop an architectural roadmap that identifies the key investments that need to be made to achieve the company's vision and IT strategy.

When developing an architectural roadmap, it is important for agile architects to be data-driven. This means using data to understand the current state of the company's technology landscape, identify trends and opportunities, and make informed decisions about resource allocation.

Architecture modeling tools can be used to create visual representations of the system's architecture. This can help to identify and communicate the key architectural components and interfaces. In addition, Portfolio management tools can be used to track and manage the investments in the architectural roadmap.


Determine your current state
Define your desired state
Conduct a gap analysis
Prioritize your actionable items
Find the best sequence
PI Plan
PI + 1
PI + 2

When

With regards to developing an Architectural Roadmap, the main difference between Intentional, Evolving & Emerging architecture is the time frame upon which you look at the roadmap:

- The Emerging Architecture details commitments and immediate objectives for a defined period, akin to a short-term roadmap. It outlines specific deliverables and milestones for the upcoming phases, potentially forecasting goals for subsequent cycles.
- The Intentional Architecture encompasses a broader view, spanning multiple years. It charts out critical milestones and necessary deliverables essential for realizing the intentional architecture's vision, focusing on long-term strategic direction.
- Evolving Architecture illustrates how the Emerging Architectures align with the broader organizational goals. It encapsulates the evolving nature of various architectural components and their role in achieving the overarching vision and objectives.

References

- **JIT-JEA Part 1: the 5 pillars** [JIT-JEA part 1]: 4.3 Just enough governance
- **Open Agile Architecture** [A-OO]: Par 6.5.2.Developing Architectural roadmap
- **SAFe Agile Architecture** [SAFe-ARCH]:
  - **SAFe Roadmap:** https://scaledagileframework.com/roadmap
- **Agile methods as part of TOGAF 10:**
  - https://pubs.opengroup.org/togaf-standard/
  - https://pubs.opengroup.org/togaf-standard/adm/chap09.html
  - https://pubs.opengroup.org/togaf-standard/digital-technology-adoption/index.html

10

Align Stakeholders (Business & IT)

Domain Model, Domain-Driven Design (DDD)

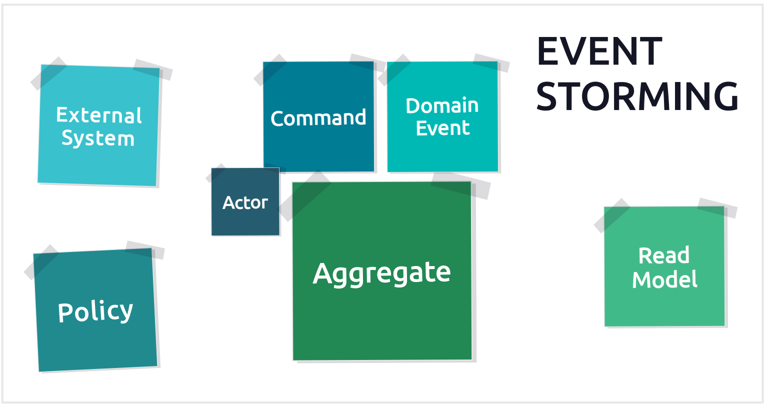
With What

During a transformation, it is essential to align business and IT. To do so, different methods, and patterns exist.

Event Storming is a workshop-based method for the collaborative exploration of complex business domains involving key stakeholders. It supports cross-discipline conversations between business (domain experts, product owners) and IT representatives (software developers). It allows to manage multiple perspectives on the same model. The goal is to maximize the learning of all the participants. At the end of the Event Storming workshop, IT representatives are ready to embrace the full power of Domain-Driven Design and microservices.

Domain-Driven Design was created by Eric Evans in 2003 following the release of his book "Tackling Complexity in the Heart of Software". DDD is a great tool for business and IT to understand each other. DDD ensures that the code and the business data model are expressed in the business language. Software design must be driven by the business. The goal is to transcribe the business intention and business needs into the software.

This design pattern helps establish a common language: A language structured around the domain model and used by all team members to connect all the activities of the team with the software. The **ubiquitous language** is a deliberate language designed to be unambiguous and on which all stakeholders agree. This language is found in every artifact manipulated by the stakeholders (UI, database, source code, documents, etc.). The concepts conveyed by the domain model are the primary means of communication. The domain model is the backbone of the ubiquitous language.



DDD delimits, through **bounded contexts**, the applicability of a particular model so that the team members have a clear and shared understanding of consistency and how it relates to other contexts. Bounded contexts simplify the architecture by separating concerns. DDD allows to divide and decouple the architecture according to the business concept to address. The domain is often modular, which makes it flexible, and easy to update with new requested changes.

When

In the intentional architecture, event storming can be used during the design phase at different levels: enterprise, domain, and sub-domain. Then in the evolving architecture or emerging architecture, event storming and DDD are essentially used at the domain or sub-domain level.

References

- **JIT-JEA Part 1** [JIT-JEA part.1]: 3. AGILE IN ARCHITECTURE FRAMEWORKS
- **Open Agile Architecture** [A-OO]:
  - Chapt. 19. Event Storming
  - Chapt. 20. Domain-Driven Design: Strategic Patterns
- **Domain Modeling:** https://scaledagileframework.com/domain-modeling/
- **Other references:**
  - Domain-Driven Design: Tackling Complexity in the Heart of Software (Eric Evans)

Authors

The authors of this Agile Architecture are all leading Architects at Capgemini.

Stefano Rossini

Chief Architect and Agile Coach

Stefano is Capgemini Italy Chief Architect expert in services architecture SOA and MSA and he is also an Agile evangelist and coach. Stefano loves both Agile and Architecture. He leads the DevOps global community and both Italian communities about Agile (Italy Agile Hub) and Architects (Italy Architects community).

Gert Helsen

Chief Architect for Financial Services

Gert is a Chief Architect at Capgemini working in the Financial Services sector. Being passionate about people and IT technology, Gert is active as a coach, mentor and a certified trainer for many architects across the Capgemini group. In addition, he leads an Innovation Service and acts as a Chief Client Architect for a strategic Financial Services client based in Europe.

Hans Van Rijs

Lead Account Architect in the Education Sector

Hans van Rijs is an all-round architect, with a solid background in software engineering and project management. He is the Lead Account Architect in the Education sector, a certified architecture trainer at the Capgemini Academy, a member of The Open Group's O-AA Work Group, and a board member of the CoP Architecture NL. Hans likes to delve into innovative technologies and agile methods and likes to share his knowledge and experience with others.

Ruud De Wit

Chief Architect for Retail

Ruud is a senior architect at Capgemini. He acts as an Client Chief Architect for clients in the Retail and Agri sector. Besides that, Ruud is an Open-Group certified trainer working for the Capgemini Academy and is a lead for the architecture domain. Ruud is driven by combining traditional and innovative technologies and development methodologies when shaping solution architectures.

Pascal Espinouse

Digital & Innovation Architect, Chief Architect

Pascal is a Chief Architect in Capgemini, specialized in Digital Transformation & Innovation. Led by passion, Pascal is a trainer and mentor within Capgemini Global Architects Community. Leader of the Architects Community of his practice, he is also part of the Core Team leading the 1500+ Architects of Capgemini France. Besides Architecture, Pascal's highest involvement relates to Sustainability and Generative AI.

Wijke Hamer

Chief Architect for Government and Public Services

Wijke Hamer is Chief Enterprise Architect at Capgemini. Since more than 25 years she has led many architecture engagements for large (international) organizations . Wijke is certified by The Open Group as trainer of Open Agile Architecture and Archimate3.2. She has achieved the final round of the global 'Women in Tech Award 2022', in the category 'Role Model', and she has given lectures about how to apply agile principles to achieve a more human centered, inclusive architecture.

About Capgemini

Capgemini is a global business and technology transformation partner, helping organizations to accelerate their dual transition to a digital and sustainable world, while creating tangible impact for enterprises and society. It is a responsible and diverse group of 340,000 team members in more than 50 countries. With its strong over 55-year heritage, Capgemini is trusted by its clients to unlock the value of technology to address the entire breadth of their business needs. It delivers end-to-end services and solutions leveraging strengths from strategy and design to engineering, all fueled by its market leading capabilities in AI, cloud and data, combined with its deep industry expertise and partner ecosystem. The Group reported 2023 global revenues of €22.5 billion.

Get the Future You Want | www.capgemini.com

MACS-Agile_19-02-2024_Priyanka Paul