



# Grounded Architecture

## Redefining IT Architecture Practice in the Digital Enterprise

Željko Obrenović

# **Grounded Architecture**

## Redefining IT Architecture Practice in the Digital Enterprise

Željko Obrenović

This book is for sale at <http://leanpub.com/groundedarchitecture>

This version was published on 2024-05-25



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2024 Željko Obrenović

# Contents

<b>1: Introduction</b> . . . . .	<b>1</b>
1.1: What Will You Learn? . . . . .	5
1.2: Key Influences . . . . .	9
1.3: Why This Book? . . . . .	10
1.4: A Part of the Bigger Picture: A Trilogy in Four Parts	11
1.5: A Bit of Personal History . . . . .	13
1.6: The Structure of the Book . . . . .	14
1.7: Stay Connected . . . . .	15
1.8: Acknowledgments . . . . .	16
<b>2: Context: Fast-Moving Global Organizations</b> . . . . .	<b>17</b>
2.1: Global Scale . . . . .	19
2.2: Multi-Dimensional Diversity . . . . .	21
2.3: Nonlinear Growth Dynamics . . . . .	24
2.4: Synergy and Transformation Pressures . . . . .	25
2.5: Decentralization and Loose Coupling . . . . .	27
2.6: Questions to Consider . . . . .	29
<b>3: Goals</b> . . . . .	<b>30</b>
3.1: Goals . . . . .	33
3.2: Questions to Consider . . . . .	35
<b>Part I: Grounded Architecture</b> . . . . .	<b>36</b>
<b>4: Grounded Architecture: Introduction</b> . . . . .	<b>37</b>

## CONTENTS

<b>5: Data Foundation . . . . .</b>	<b>40</b>
5.1: Requirements For A Data Foundation . . . . .	43
5.2: Examples of Data Foundation Sources and Tools . . . . .	45
5.3: Building Data Foundation . . . . .	52
5.4: Using Architecture Data Foundation . . . . .	54
5.5: Appendix: Examples of Insights From Source Code Analyses . . . . .	55
5.6: Questions to Consider . . . . .	59
<b>6: People Foundation . . . . .</b>	<b>60</b>
6.1: Background: Central vs. Federated Architecture Function . . . . .	63
6.2: The Hybrid Model . . . . .	65
6.3: Building People Foundation . . . . .	69
6.4: Questions to Consider . . . . .	71
<b>7: Architecture Activities Platform . . . . .</b>	<b>72</b>
7.1: Examples of Architecture Activities . . . . .	75
7.2: Operating Model . . . . .	77
7.3: Architecture Decision Policy . . . . .	79
7.4: Distributing Decisions, Autonomy, and Alignment . . . . .	81
7.5: Rules of Engagement . . . . .	83
7.6: Questions to Consider . . . . .	86
<b>8: Value of Grounded Architecture Structure . . . . .</b>	<b>87</b>
8.1: Executing at Scale . . . . .	89
8.2: Adaptivity . . . . .	91
8.3: Improving the Quality of Decision-Making with Data . . . . .	93
8.4: Maximizing Organizational Alignment . . . . .	95
8.5: Maximizing Organizational Learning . . . . .	97
8.6: Questions to Consider . . . . .	99
<b>Part II: Being Architect . . . . .</b>	<b>100</b>
<b>9: Being Architect: Introduction . . . . .</b>	<b>101</b>

## CONTENTS

<b>10: Architects as Superglue . . . . .</b>	<b>103</b>
10.1: Supergluing in Action: Reducing Tension among Business Functions, Product, Technology, Organization . . . . .	106
10.2: Superglue Abilities . . . . .	110
10.3: Questions to Consider . . . . .	114
<b>11: Skills . . . . .</b>	<b>115</b>
11.1: Hard Skills . . . . .	117
11.2: Soft Skills . . . . .	119
11.3: Product Development Skills . . . . .	121
11.4: Business Skills . . . . .	122
11.5: Decision-Making Skills . . . . .	123
11.6: Questions to Consider . . . . .	124
<b>12: Impact . . . . .</b>	<b>125</b>
12.1: Pillars of Impact . . . . .	127
12.2: Questions to Consider . . . . .	131
<b>13: Leadership . . . . .</b>	<b>132</b>
13.1: David Marquet's Work: The Leader-Leader Model . . . . .	134
13.2: Netflix's Valued Behaviors: Leadership Behaviors . . . . .	137
13.3: Questions to Consider . . . . .	142
<b>14: Architects' Career Paths: Raising the Bar . . . . .</b>	<b>143</b>
14.1: Typical Architect's Career Paths . . . . .	145
14.2: Hiring Architects . . . . .	147
14.3: Questions to Consider . . . . .	150
<b>Part III: Doing Architecture: Inspirations . . . . .</b>	<b>151</b>
<b>15: Doing Architecture: Introduction . . . . .</b>	<b>152</b>
<b>16: The Culture Map: Architects' Culture Compass . . . . .</b>	<b>155</b>

## CONTENTS

16.1: Communicating . . . . .	158
16.2: Evaluating . . . . .	160
16.3: Persuading . . . . .	162
16.4: Leading . . . . .	164
16.5: Deciding . . . . .	166
16.6: Trusting . . . . .	168
16.7: Disagreeing . . . . .	170
16.8: Scheduling . . . . .	172
16.9: Rules . . . . .	174
16.10: Questions to Consider . . . . .	175
<b>17: Managing Organizational Complexity: Six Simple Rules</b> . . . . .	<b>176</b>
17.1: Background: Limitations of Hard and Soft Management Approaches . . . . .	178
17.2: Six Simple Rules Overview . . . . .	180
17.3: Rule 1: Understand What Your People Do . . . . .	182
17.4: Rule 2: Reinforce Integrators . . . . .	184
17.5: Rule 3: Increase the Total Quantity of Power . . . . .	186
17.6: Rule 4: Increase Reciprocity . . . . .	188
17.7: Rule 5: Extend the Shadow of the Future . . . . .	190
17.8: Rule 6: Reward Those Who Cooperate . . . . .	192
17.9: Questions to Consider . . . . .	194
<b>18: Understanding Product Development</b> . . . . .	<b>195</b>
18.1: The Build Trap . . . . .	199
18.2: The Discipline of Market Leader . . . . .	206
18.3: Product Operations . . . . .	210
18.4: Questions to Consider . . . . .	215
<b>19: Architecture Governance: Nudge, Taxation, Mandates</b> . . . . .	<b>216</b>
19.1: Nudging . . . . .	219
19.2: Taxation (Economic Incentives) . . . . .	222
19.3: Mandates and Bans . . . . .	224
19.4: Questions to Consider . . . . .	226
<b>20: Economic Modeling: ROI and Financial Options</b> . . . . .	<b>227</b>

## CONTENTS

20.1: The Return-on-Investment Metaphor . . . . .	230
20.2: The Financial Options Metaphor . . . . .	232
20.3: A Communication Framework . . . . .	234
20.4: Questions to Consider . . . . .	238
<b>21: Decision Intelligence in IT Architecture . . . . .</b>	<b>239</b>
21.1: Basics of Decision-Making . . . . .	242
21.2: Preparing for Making Decisions . . . . .	246
21.3: Decision-Making Complexity . . . . .	251
21.4: Decision-Making With Data and Tools . . . . .	256
21.5: Questions to Consider . . . . .	259
<b>22: The Human Side of Decision-Making . . . . .</b>	<b>260</b>
22.1: Biases and Limitations . . . . .	262
22.2: Decisiveness . . . . .	275
22.3: Intuition . . . . .	277
22.4: Group Decision-Making Dynamics . . . . .	281
22.5: Questions to Consider . . . . .	287
<b>Part IV: Wrapping Up . . . . .</b>	<b>288</b>
<b>23: Summary . . . . .</b>	<b>289</b>
<b>24: Cheat Sheet . . . . .</b>	<b>293</b>
24.1: Introductions . . . . .	294
24.2: Grounded Architecture: Introduction . . . . .	295
24.3: Being Architect: Introduction . . . . .	297
24.4: Doing Architecture: Introduction . . . . .	299
<b>Part V: To Probe Further . . . . .</b>	<b>302</b>
<b>25: Bookshelf . . . . .</b>	<b>303</b>
25.1: Introduction . . . . .	304
25.2: Career Development . . . . .	307

## CONTENTS

25.3: Hard Skills . . . . .	308
25.4: Soft Skills . . . . .	314
25.5: Organization and Processes . . . . .	319
25.6: Business, Product, Strategy . . . . .	321
<b>26: Tools . . . . .</b>	<b>323</b>
<b>27: Favorite Quotes . . . . .</b>	<b>326</b>
27.1: People, People, People . . . . .	327
27.2: Changing Role of Architecture . . . . .	328
27.3: Complexity . . . . .	329
27.4: Other . . . . .	330
<b>Part VI: Appendix . . . . .</b>	<b>331</b>
<b>28: Appendix Overview . . . . .</b>	<b>332</b>
<b>29: ISO 25010 Standard . . . . .</b>	<b>334</b>
29.1: Overview . . . . .	335
29.2: Maintainability . . . . .	336
29.3: Security . . . . .	338
29.4: Performance Efficiency . . . . .	340
29.5: Reliability . . . . .	342
<b>30: Cloud Design Patterns . . . . .</b>	<b>344</b>
30.1: Overview . . . . .	345
30.2: Performance and Scalability . . . . .	346
30.3: Resiliency . . . . .	347
30.4: Messaging . . . . .	348
30.5: Management and Monitoring . . . . .	349
30.6: Security . . . . .	350
30.7: Other Patterns . . . . .	351
<b>31: High Performing Technology Organizations . . . . .</b>	<b>352</b>
31.1: Overview . . . . .	353
31.2: Four Key Metrics . . . . .	354

31.3: Practices . . . . .	355
---------------------------	-----

# 1: Introduction



image by fda54 from pixabay

**IN THIS SECTION, YOU WILL:** Understand what this book is about and how to use it.

**KEY POINTS:**

- This book will share my approach to running an IT architecture practice in larger organizations based on my experience at AVIV Group, eBay Classifieds, and Adevinta. I call this approach “Grounded Architecture”—architecture with strong foundations and deep roots.
- Prioritizing people interactions and data over processes and tools, Grounded Architecture aims to connect architecture practice to all organizational levels as an antidote to the “ivory tower” architecture.
- I also explain my motivation to write this book.

This book shares my approach to running an IT architecture practice in larger organizations based on my experience at AVIV Group, eBay Classifieds, and Adevinta. I call this approach “**Grounded Architecture**”—architecture with **strong foundations** and **deep roots**. Prioritizing people interactions and data over processes and tools, Grounded Architecture aims to connect architecture practice to all organizational parts and levels, serving as an antidote to the “ivory tower” architecture.

I use the name Grounded Architecture, drawing a parallel with the **Grounded Theory**<sup>1</sup> methodology. Grounded Theory emphasizes grounding the development of theories and concepts directly in the empirical data collected from real-world observations and experiences. The term “grounded” refers to the idea that the theory is firmly rooted in the data and is not based on preconceived hypotheses or existing theoretical frameworks. The theory is not developed in advance but emerges from the data as it is analyzed.

Similarly, I wanted to create an approach to architecture practice that is more **empirical and firmly rooted in data**. An architecture

---

<sup>1</sup>[https://en.wikipedia.org/wiki/Grounded\\_theory](https://en.wikipedia.org/wiki/Grounded_theory)

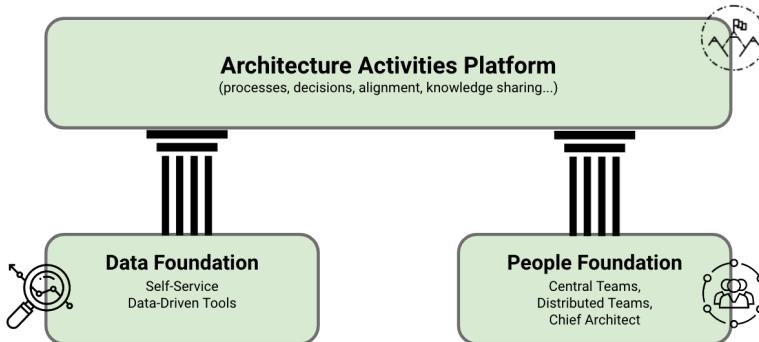
practice that is not developed in advance based only on abstract principles and frameworks but also emerges and adapts based on feedback from practice.

This book introduces Grounded Architecture as an IT architecture practice with two main parts:

- **Structure**, elements you need to have to run Grounded Architecture practice, and
- **Guiding Principles**, pieces of advice that can help put the ideas of Grounded Architecture into practice.

Figure 1 shows the structure of the Grounded Architecture consisting of three elements:

- **Data Foundation**,
- **People Foundation**,
- **Architecture Activities Platform**.



**Figure 1:** The structure of Grounded Architecture.

The **Data Foundation** ensures that architects can make data-informed decisions based on a real-time and complete overview of the organization's technology landscape.

The **People Foundation** is another essential element of Grounded Architecture. A strong network of people doing architecture across

the organization is crucial to ensure that architecture function has any tangible impact.

Lastly, the *Architecture Activities Platform* defines a set of processes and agreements enabling architects to do everything architecture typically does, leveraging data and People Foundations to create a data-informed, organization-wide impact.

As a part of my work on Grounded Architecture, I also provide several guiding principles and tools that I found helpful to introduce the ideas of Grounded Architecture in practice. I grouped these resources into two parts:

- **Being an Architect**, where I introduce principles that generalize my view on what it means to be an architect in practice:
  - Architects as Superglue
  - Skills
  - Impact
  - Leadership
  - Architects' Career Paths
- **Doing Architecture: Inspirations**, where I introduce several resources that I use as inspiration for running the Grounded Architecture practice in complex organizations:
  - Culture Map: Architects' Culture Mindfield Compass
  - Managing Organization Complexity: Six Simple Rules
  - Product Development and The Build Trap
  - Architecture Governance: Mandates, Taxation, Nudge
  - Economic Modeling: ROI and Financial Options
  - Decision Intelligence in IT Architecture
  - The Human Side of Decision-Making

The rest of this book will explain in detail the Grounded Architecture approach. In this section, I want to tell a few things about my motivation to write this book.

## 1.1: What Will You Learn?

The three parts of the book (Structure, Being an Architect, and Doing Architecture) correspond to the aspects of work of **Chief Architects or Heads of Architecture** that need to set up and run modern IT architecture practices:

- Create organizational and technical **structures** to support IT architecture work,
- Define **IT architecture roles** and responsibilities, skills, and career paths,
- Operate **effective IT architecture practice** in complex multicultural organizations.

### 1.1.1: General Philosophy

I approach designing an architecture practice in complex organizations as an **art of cooking**. Modern **IT architecture practice** is **highly adaptive**, and applying complex frameworks and models is not practical and effective. You cannot easily copy an approach from one organization to another. Consequently, I have organized this book into **modules you can mix together in your context**, similar to how a chef may use ingredients in a recipe to create specific outcomes. I share some of the lessons and tips on “cooking,” but you must find your way in your “kitchen.” You may reuse some of the ingredients and recipes from this book. You may not need some of this book’s ingredients. And you will need many more from other sources.

When I start an architecture practice, I view myself as a chef who joins a new restaurant and brings their favorite spices to enhance local dishes. I incorporate **core preferred elements and foundational frameworks** like essential culinary spices and tools.

However, the true essence of the architecture, akin to **the unique flavor of a dish, is derived from local factors**. In an organization, these local elements are the people: the **in-house talent and culture** that shape the enterprise's distinct identity.



image by istock

While basic structures and practices may be consistent across various business scenarios, the most crucial aspects—like fresh, local ingredients in cooking—need to be locally sourced. Skills, experiences, and insights of the organization's staff play a pivotal role in **customizing and refining it to meet specific business needs and goals**. They make the architecture truly tailored and effective.

### 1.1.2: Format

I have organized my lessons and insights in a form that, if you recognize the problems and are inspired by solutions, could use as a **high-level “playbook”** about how to work as an architect or run an architecture practice. I also provide more concrete tips on each

discussed topic, finishing each section with questions you should consider when addressing these topics.

### **1.1.3: Content**

This book is **not technical**. We will not discuss the details of public cloud design patterns, security, reliability, how to optimize computer loads, or select the proper data storage. As a modern architect, you will need these skills, of course, but there are already many great resources for them. This book is about **expanding your horizons** to apply your technical skills in complex organizations. Or to broaden your horizons as a head or manager of architects, to organize and support architects to use their technical skills more effectively as a team.

### **1.1.4: Is This A Proven Method?**

Like with many similar books, you may be disappointed if you are looking for a scientifically proven “method” of running a modern architecture practice. This book is **personal and opinionated**, building on my daily experiences as an architect and head of an architecture practice. While subjective, I believe this book can provide **valuable insights** for IT architects, their managers, and people working with architects. I have successfully **applied my approach in three different companies**, which gives it some generality and repeatability.

I invite others to share the lessons they have learned similarly. Even if opinionated and limited in scope, such practical reflections based on concrete examples have much more value for practitioners than abstract debates, formal methods, or academic analyses.

### **1.1.5: Who Should Read This Book?**

When writing this book, I had a **broad audience** in my mind. The article should be helpful to both technical and non-technical people. The book can help IT architects to better understand their value and place in a broader organization. I also hope the articles show the wider audience the benefits of staying close to and well-connected with architects.

## 1.2: Key Influences

The Grounded Architecture approach also builds on many ideas others have successfully used. Gregor Hohpe's **Architecture Elevator**<sup>2</sup> view of architecture has heavily inspired my work. In many ways, my work reflects the lessons learned from implementing Gregor's ideas in practice. Gregor described modern architects' functions as aligning organization and technology, reducing friction, and charting transformation journeys. Such modern architects ride the Architect Elevator from the penthouse, where the business strategy is set, to the engine room, where engineers implement enabling technologies.

In my quest to define modern architectural roles, I used Staff+ Engineering jobs as an inspiration for the development of architects. Tanya Reilly's book **The Staff Engineer's Path**<sup>3</sup> and Will Larson's book **Staff Engineer: Leadership beyond the management track**<sup>4</sup> are helpful guides in defining the responsibilities of modern architects. Overall, the Staff-plus engineering roles provide excellent examples for the development of architects.

Many other sources have influenced my work. Some of them you can find in the [Bookshelf](#) section.

---

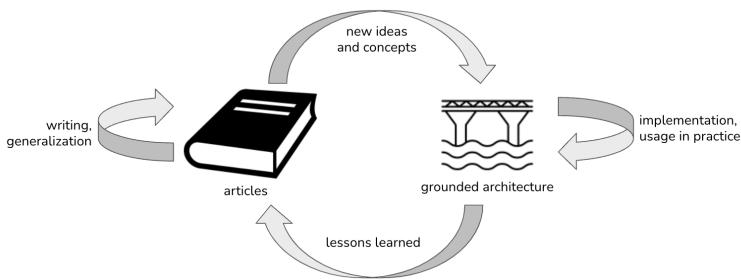
<sup>2</sup><https://architectelevator.com/>

<sup>3</sup><https://www.oreilly.com/library/view/the-staff-engineers/9781098118723/>

<sup>4</sup><https://staffeng.com/guides/staff-archetypes/>

## 1.3: Why This Book?

This book **generalizes my experiences** in a written form. I have written these articles for several reasons. Firstly, the act of writing helps me to clarify and improve my ideas (Figure 2). As Gregor Hohpe once noted, I free up some brain cells to learn new things with every sentence I write.



**Figure 2:** Writing a book helped me organize ideas, obtain new insights, improve principles and tools, and share the lessons learned.

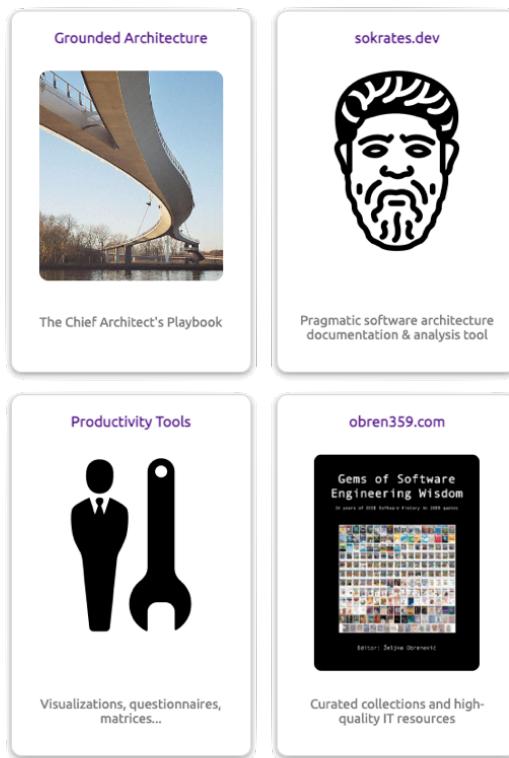
I also needed this book for the **education of architects** and to **increase awareness about modern architecture practices** in organizations I worked in. Having written content can significantly help to spread the message. As nicely described by Hohpe written word has distinct advantages over the spoken word:

- it scales: you can address a broad audience without gathering them all in one (virtual) room at the same time
- it's fast to process: people read 2-3 times faster than they can listen
- it can be easily searched or versioned.

Lastly, by generalizing and putting my experiences on paper, I aim to create more **usable materials to help others** in similar situations. I also expect helpful feedback from a broader community.

## 1.4: A Part of the Bigger Picture: A Trilogy in Four Parts

This book is a part of the **collection of open-source tools and resources**<sup>5</sup> I have built in the past ten years to help me in architectural work (Figure 3).



**Figure 3:** Grounded Architecture is a part of the collection of open-source tools and resources I have built in the past ten years to help me in architectural work.

<sup>5</sup><https://obren.io/>

The other resources include:

- **Sokrates**<sup>6</sup>: an open-source polyglot code analysis tool. Sokrates provides a pragmatic, inexpensive way to extract rich data from source code repositories. Sokrates can help you understand your code by making the size, complexity, and coupling of software, people interactions, and team topologies visible.
- **Productivity Tools**<sup>7</sup>: A collection of more than 100 online tools I built to help me in my daily work as an architect.
- **359° Overview of Tech Trends**<sup>8</sup> is my collection of knowledge resources with podcasts and videos from over 20 authoritative, high-quality sources (IEEE, ACM, GOTO Conf, SE Radio, Martin Fowler's site, Ph.D. Theses). Architects need to learn fast, and finding good knowledge sources is difficult.

You can find more details about these tools on my homepage [obren.io](https://obren.io)<sup>9</sup>.

---

<sup>6</sup><https://sokrates.dev>

<sup>7</sup><https://obren.io/tools>

<sup>8</sup><https://www.obren359.com/>

<sup>9</sup><https://obren.io/>

## 1.5: A Bit of Personal History

The work presented in this book builds on **several years of my experience**. Most of this work originates from my current work as a **Chief Architect** at AVIV Group and previous works as a **Principal Architect** for eBay Classifieds and Adevinta.

Another vital part of my experience that shaped this book was my earlier experience as a **consultant and analyst** at the **Software Improvement Group**<sup>10</sup>. I've learned the value and pragmatics of data-informed decision-making. As a spin-off of this work, I've also built a tool called **Sokrates**<sup>11</sup>, which enables efficient and pragmatic extraction of data about technology and organization from source code. This work has directly influenced my view of the architecture **Data Foundation**.

My experience as a **CTO** of **Incision**<sup>12</sup>, a startup, has helped me better understand the challenges of creating and running an IT organization.

My experience as a **researcher** at **Dutch Center for Computer Science and Mathematics (CWI)**<sup>13</sup> and **Eindhoven Technical University (TU/e)**<sup>14</sup> provided me with a valuable background to do rigorous data analyses and research. From this research period, I want to highlight a collection of essays, **Design Instability**<sup>15</sup>, that I co-authored with Erik Stolterman, where we connected experiences of designing/architecting in three disciplines: classical design, UX design, and software engineering. This work has helped me better relate to and learn from non-technical fields.

Lastly, my hands-on experience as a **software developer** has proven invaluable for my work as an architect.

---

<sup>10</sup><https://www.softwareimprovementgroup.com/>

<sup>11</sup>[sokrates.dev](http://sokrates.dev)

<sup>12</sup><https://incision.care>

<sup>13</sup><https://www.cwi.nl/en/>

<sup>14</sup><https://www.tue.nl/en/>

<sup>15</sup><https://design-instability.com/>

## 1.6: The Structure of the Book

I have organized the book into several main parts. In the introductory part, I describe the context in which my ideas have developed.

In the second part, I discuss the Grounded Architecture structure describing its three elements: the Data Foundation, the People Foundation, and the Architecture Activities Platform.

In the third part, I discuss the Guiding Principles of Grounded Architecture, grouped into two sections: Being an Architect and Doing Architecture.

I conclude with a summary and pointers to external resources for those who want to explore more.

The [bookshelf](#) section shows many other books and resources that have influenced my approach and ideas.

I invite you to read this book from **beginning to end**, following the progression from data and basic structures to management and organizational topics. However, you can also **browse the text** and start reading whatever interests you. I use many **illustrations** to create easy-to-remember pictures that you can associate with discussed topics, making the book usable across your organization as a **coffee table book**, serving as an inspiration, or sparking discussions.

## 1.7: Stay Connected

You can find additional resources online at:

- <https://grounded-architecture.io><sup>16</sup>

Feel free to follow me on LinkedIn to see what I am up to:

- <https://www.linkedin.com/in/zeljkoobrenovic>

---

<sup>16</sup><https://grounded-architecture.io/>

## 1.8: Acknowledgments

Thank all AVIV Group's Architecture Center of Excellence members and eBay Classifieds Virtual Architecture Team (VAT) members who gave me invaluable feedback and discussions. Lastly, thank Peter Maas and Brent McLean for sponsoring and pushing for developing data-informed architecture in our organizations.

The cover image is [a photo of Nesciobrug<sup>17</sup>](#). Credit: the botster, CC BY-SA 2.0, via Wikimedia Commons.



image by henk monster cc by 3 0 via wikimedia commons

---

<sup>17</sup>[https://commons.wikimedia.org/wiki/File:Nesciobrug\\_4.jpg](https://commons.wikimedia.org/wiki/File:Nesciobrug_4.jpg)

## **2: Context: Fast-Moving Global Organizations**



image by paul brennan from pixabay

**IN THIS SECTION, YOU WILL:** Understand the context in which the ideas in this book developed.

**KEY POINTS:**

- To better understand any idea or solution, it is crucial to understand the context in which this idea developed.
- The Grounded Architecture approach has evolved in the context of global, loosely coupled organizations that are diverse, with nonlinear growth dynamics, and under transformation pressures.

My approach to creating and running an architecture function is not an abstract idea. Instead, it is the generalization of lessons learned while solving specific problems in a particular context.

To better understand any idea or solution, it is crucial to understand the context in which these ideas developed. I base my views on my experiences as a Chief Architect at AVIV Group and a Principal Architect at eBay Classifieds and Adevinta. In this section, I discuss critical characteristics of the organizational context that have had an impact on my definition of the Grounded Architecture approach:

- **Global scale:** operating across multiple countries and continents with millions of users.
- **Multi-dimensional diversity:** the organizations I worked in were diverse, including customer base, workforce, business models, team topologies, and technology stacks.
- **Nonlinear growth dynamics:** besides organic growth, big organizations change their portfolio through mergers and acquisitions of new businesses or divestments.
- **Synergies and transformation pressures:** big organizations do not want just to be big. They want to exploit the benefits of the economies of scale and reduce duplication of efforts.
- **Decentralized, loosely-coupled organizational units:** organizational units have significant autonomy.

## 2.1: Global Scale

I have developed my approach in genuinely global and multicultural organizations on a massive scale:

- Operating across many geographies, cultures, and languages,
- Serving millions of users each day,
- Working with thousands of software developers in hundreds of product and development teams,
- Implementing systems with hundreds of millions of lines of source code.



image by pete linforth from pixabay

The global scale introduces several compelling opportunities for organizations. The global scale can increase organizational effectiveness due to the possibility of **reducing duplication of effort** by centralizing shared activities. Second, the global scale enables leveraging **economies of scale** by achieving cost advantages, such as lowering unit prices of used technologies. Furthermore, the global scale can increase **business resilience and flexibility**, possibly compensating for negative local market changes with global

resources. Global organizations also have a **bigger talent pool** to support local or international efforts. Lastly, such organizations have significant resources to invest in to support nonlinear growth through mergers and acquisitions (M&As).

The global and massive scale brings many challenges. It leads to **high organizational complexity**, with thousands of possible communication channels among the organization's units. Global scale means having a **complex technology landscape** with many services and interconnections. An immense talent pool also means **high costs** for the workforce continuously. And such organizations also have constantly high costs of computing resources due to the need to serve many customers twenty-four hours a day, seven days a week, all the time. Operating in many places also increases the **complexity of running operations** with high and variable customer demands. With many data centers, services, and applications, global organizations have a **vast attack surface**, with many points on the boundary of their systems where an attacker can try to enter, cause an effect on, or extract data. Lastly, any manual process, such as diagram drawing to create an overview of the organizational or technology landscape, is limited due to scale.

Balancing opportunities and challenges on a global scale has been one of the most challenging and rewarding aspects of my architectural work.

## 2.2: Multi-Dimensional Diversity

The organizations I worked in were very diverse across several dimensions:

- **Cultures:** different (local and remote) workforce and customers,
- **Organization:** different sizes, complexity, and styles of organizational units,
- **Product:** diverse sets of features covering many markets and different customer segments,
- **IT Architecture:** legacy and modern approaches mixed, and
- **Technology:** dozens of programming languages and thousands of third party libraries, frameworks, and services.



image by simon from pixabay

Organization-wise, I needed to work with many units that differed in multiple ways:

- **Unit size:** some units had hundreds of people, and some had only a dozen.
- **Team topologies:** some organizations had one team, while other units organized teams hierarchically,
- **Position of architecture:** with some organizations having local architecture teams and local lead architects, smaller units have their team members conducting architecture activities in addition to other responsibilities.

Architecture-wise, we needed to work with many styles in active production systems, ranging from **legacy monolith applications** to complex modern **microservice and serverless ecosystems**. Each part of the organization has a different history and a different legacy.

And our technology covered almost any of the mainstream stacks. The technical infrastructure included several public cloud providers (AWS, GCP, Azure) and custom-built private data centers. The systems also employed diverse application technologies, such as:

- **Database technologies** (e.g., MySQL, PostgreSQL, MongoDB, Cassandra, AWS RDS...).
- **Backend programming languages** (e.g., Java, C#, Go, Scala, PHP, Node.js, Kotlin...).
- **Mobile app programming languages** (e.g., Swift, Objective-C, Java, Kotlin, or Flutter/Dart...).
- **Frontend programming languages and frameworks** (e.g., React, Android, AndroidJS, Vue, or jQuery).

From an architectural perspective, diversity offers several opportunities. It can **increase technology innovation** due to a diverse

workforce and the possibility of creatively exploring more technologies and tools. And it can **help address various implementation needs** better by choosing from a more extensive and diverse pool of resources (selecting the best tool for the job).

Diversity also poses several challenges. High diversity **increases overall system landscape complexity** and the cognitive load of teams who must master many different topics simultaneously. Diversity can also **reduce flexibility** and reorganization possibilities as expertise is split among many domains and technologies. And the variety of technology stacks also may lead to **higher technical debt** due to many legacy components in many (outdated) technologies.

From an architectural perspective, diversity is an excellent source of new possibilities but comes with challenges in controlling complexity.

## 2.3: Nonlinear Growth Dynamics

Complex organizations like the one I have worked in are frequently very dynamic. Such organizations grow (or shrink) and reorganize often and significantly. They change both organically and inorganically. **Organic growth** is internal growth the company sees from its operations. **Inorganic change** comes from buying other businesses, opening new locations, or divesting.



image by pixels from pixabay

Nonlinear growth, in particular, may be helpful in several scenarios. It can **quickly increase the customer base** or add new market segments. And such change can also **speed up innovation** due to the acquisition of new technologies or services.

But nonlinear growth dynamics have significant influences on architectural activities. The sudden addition of new companies **increases organizational complexity** with many new units. Obtaining a new company also **adds new technology and engineering units** with new processes and technology stacks. And nonlinear dynamics also **require complex architecture** to stay flexible if the organization decides to divest a part of the organization.

## 2.4: Synergy and Transformation Pressures

Complex organizations do not just grow. Instead, they want to be more efficient and leverage economies of scale, cost synergies, or increase capacity for innovation. Our investors expect us to transform to be **more than a sum of our original parts**.



image by mustangjoe from pixabay

Pressure for synergies and transformations can provide several opportunities. Synergies lead to **cost reductions** and less duplication. Resulting cost reductions can **accelerate innovation** due to more resources freed after synergies. Creating synergic components gives us more possibilities for **reuse and sharing**. And well-executed transformations can create **more efficiencies and lower unit costs**.

Pressure to be more synergic and efficient has its challenges. **Up-front investment** is needed to gain any benefit. Such investment typically brings **high risks**. Moreover, teams are often pressured to **perform** and deliver excellent short-term results while significantly **transforming**. The productivity of some units may (temporarily) drop due to the need to balance efforts between transformation activities and other work. And after transformations, the **complexity of the organization and technology landscape** may increase due to more dependencies, e.g., reusing central services.

Synergies and transformation pressures can lead to high expectations and pressure that complicate regular architecture work. On the positive side, such forces can create many new opportunities.

## 2.5: Decentralization and Loose Coupling

Researcher Karl Weick developed the concepts of tight and loose coupling to describe the organizational structure first in educational institutions and later applied to diverse businesses. According to Weick, a **tightly coupled organization** has mutually understood rules enforced by *inspection and feedback* systems. In tightly coupled organizations, management can more directly coordinate different departments' activities according to a central strategy.

In a **loosely coupled organization**, some of the elements of a tightly coupled organization are absent. Employees have **more autonomy**, and different departments may operate with **little coordination**.



image by shire777 from pixabay

Due to historical and strategic reasons, most organizational units I worked with were loosely coupled. Our companies frequently grow

through acquisitions of companies in different marketplaces. The business strategies also frequently promote a more independent evolution of local units to address local market needs better and faster. Such units often have a high level of autonomy, frequently with their development teams and sometimes with local CFOs, CMOs, or CEOs.

Loose coupling offers several advantages:

- **Higher flexibility:** units can keep developing independently, addressing specific needs without synchronizing with other units.
- **High development speed / faster time-to-market:** fewer dependencies make it much easier for marketplaces to change and evolve their products for local needs.
- **Innovation:** possibilities to quickly explore ideas in smaller contexts.

Loose coupling also has several challenges:

- **Duplication of effort:** while local market needs differ, there is frequently a significant overlap in product features and technology. This overlap leads to duplication of effort as each marketplace creates solutions for the same problems.
- **Increased accidental diversity:** limited synchronization offers flexibility but may lead to significantly different design and technology choices for the same problem, making it challenging to consolidate solutions, move people between teams, or benefit from the economy of scale.
- **Limited possibilities for central control:** due to fewer dependencies and different goals, it is more difficult to introduce changes across the board.

Loose coupling is architecture-wise an interesting challenge as it frequently leads to a conflict between global alignment and control and local autonomy.

## 2.6: Questions to Consider

To better understand any idea or solution, it is crucial to understand the context in which these ideas developed. When using ideas from this book, ask yourself how your organizational context differs from mine:

- *What are the unique characteristics of your organizational context?*
- *What is the scale of your organization? How it affects architecture function?*
- *How diverse is your organization?*
- *What are the growth dynamics of your organization?*
- *Are you experiencing synergy and transformation pressures?*
- *How (de)centralized is your organization?*

# 3: Goals



image by istock

**IN THIS SECTION, YOU WILL:** Understand the requirements I identified for an architecture function in complex organizations.

**KEY POINTS:**

- I identified the following needs that an architecture function should support: Executing At Scale, Adaptivity, Improving the Quality of Decision-Making with Data, and Maximizing Organizational Alignment & Learning.

Grounded Architecture was born as a response to the challenges discussed in the [previous section](#). Considering the scale and complexity of our organizational context, we needed to revise conventional approaches to architecture that rely on manual processes.

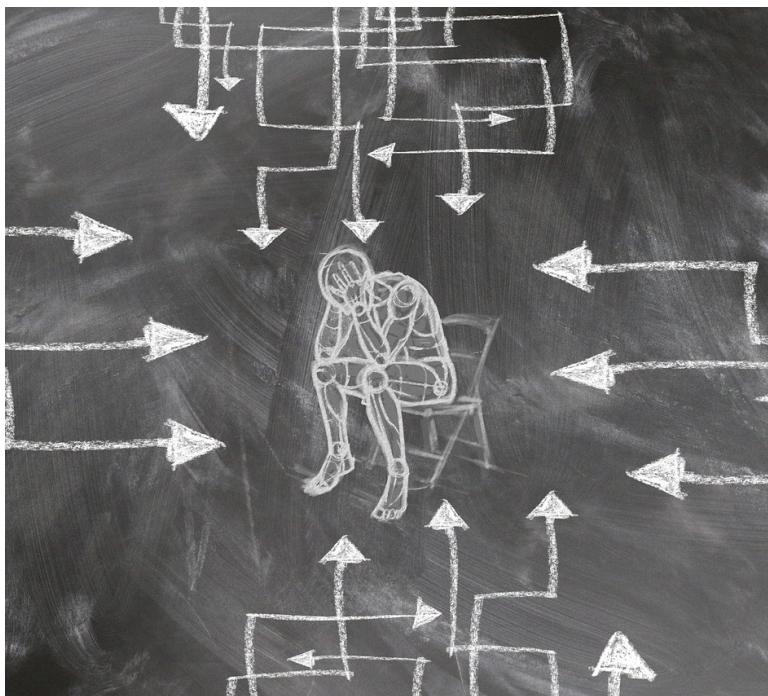


image by gerd altmann from pixabay

## 3.1: Goals

More specifically, I identified the following needs that an architecture function should support in complex organizations.

### 3.1.1: Goal 1: Executing At Scale

We needed to find a way to **support all** of the hundreds of teams, and thousands of projects, with significant complexity and diversity.

### 3.1.2: Goal 2: Adaptivity

Significant organic and inorganic changes are frequent and expected. The architecture function must **adapt quickly to stay relevant** in new contexts.

### 3.1.3: Goal 3: Increasing Quality of Decisions with Data

Intuition does not work at scale. We need tools and mechanisms to make a decision process more data-informed and **less dependent on opinions**. Furthermore, complex organizations' cultural and organizational diversity makes opinion-driven decision-making processes highly ineffective.

### 3.1.4: Goal 4: Maximizing Organizational Alignment

Misalignment is a natural state in a global, diverse, fast-moving organization. The architecture function should be a **cohesive factor**

**in minimizing such misalignments.** Otherwise, architecture may accelerate the creation of chaos.

### **3.1.5: Goal 5: Maximizing Organizational Learning**

In complex organizations with lots of effort needed to maintain legacy systems, learning and following new technology developments takes work. Architecture should **help organizations learn quickly**, stay up-to-date with emerging technologies and industry trends, and recommend technology upgrades.

## 3.2: Questions to Consider

Knowing what goals architecture practice needs to support in your organization is crucial to define structures and measure your impact. Some of the plans may be universally applicable. Others may be unique to your context. Ask yourself the following questions:

- *What is the scale of your architecture function? Does your scale require special measures to ensure your architecture practice efficient operations?*
- *What are the key decisions you need to make? Do you have the data to base your decisions?*
- *How aligned are units in your organizations? How much friction is there? How can architecture function help?*
- *How much is your organization learning? How is the learning supported?*
- *How stable is your organization? How likely is it that significant changes will occur in your organization?*

# **Part I: Grounded Architecture**

# 4: Grounded Architecture: Introduction



image by ichigo121212 from pixabay

**IN THIS SECTION, YOU WILL:** Get an overview of the Grounded Architecture structure: Data Foundation, People Foundation, and Architecture Activities Platform.

**KEY POINTS:**

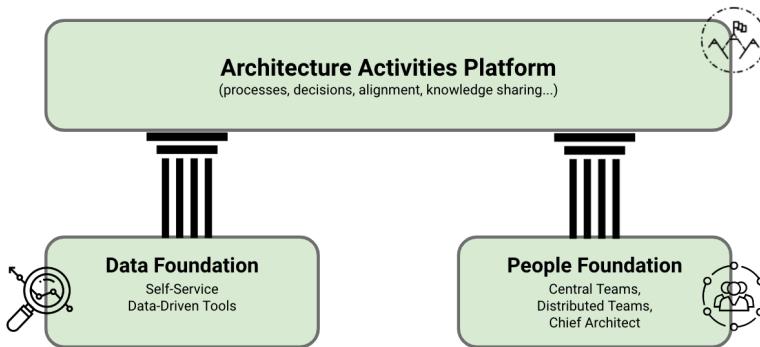
- I introduce three elements of Grounded Architecture: The Data Foundation, The People Foundation, and The Architecture Activities Platform as an approach to setting organizational structures for a modern IT architecture practice.
- Prioritizing people interactions and data over processes and tools, Grounded Architecture aims to connect architecture practice to all organizational levels as an antidote to the “ivory tower” architecture.

In this part, I will briefly introduce the structure of Grounded Architecture. I chose the term “Grounded Architecture” to highlight that the primary goal of my approach is **avoiding having an “ivory tower” architecture function** disconnected from the organization, which, in a **fast-moving, global, and diverse setting**, is a real danger. In other words, I wanted to create an architectural function that is **well-grounded in the organization**.

Prioritizing people interactions and data over processes and tools, Grounded Architecture aims to **connect architecture practice to all organizational parts and levels**, serving as an antidote to the “ivory tower” architecture.

Grounded Architecture as an approach to setting organizational structures for architecture practice has three elements:

- The Data Foundation,
- The People Foundation,
- The Architecture Activities Platform.



*Figure 1: The structure of Grounded Architecture.*

The *Data Foundation* is a system of tools and resources that enables architects to make **data-informed decisions** based on a real-time and complete overview of the organization's technology landscape. The [Data Foundation section](#) provides more details.

The *People Foundation* is a **network of people** doing architecture across the organization. This Pillar is crucial to ensure that architecture function has any **tangible impact**. As noted by Gregor Hohpe, to transform an organization, you do not need to solve mathematical equations; you need to move people. The [People Foundation section](#) provides more details.

Lastly, the *Architecture Activities Platform* defines a set of **processes and agreements** enabling architects to do everything that architecture typically does, leveraging data and People Foundations to create a data-informed, organization-wide impact. The [Architecture Activities Platform section](#) provides more details on the Architecture Activities Platform.

The Architecture Activities Platform is **only valid with the healthy Data and People Foundations**. Without data and people connections, an Architecture Activities Platform becomes an ivory tower institution, generating opinion-based decisions disconnected from reality.

# 5: Data Foundation

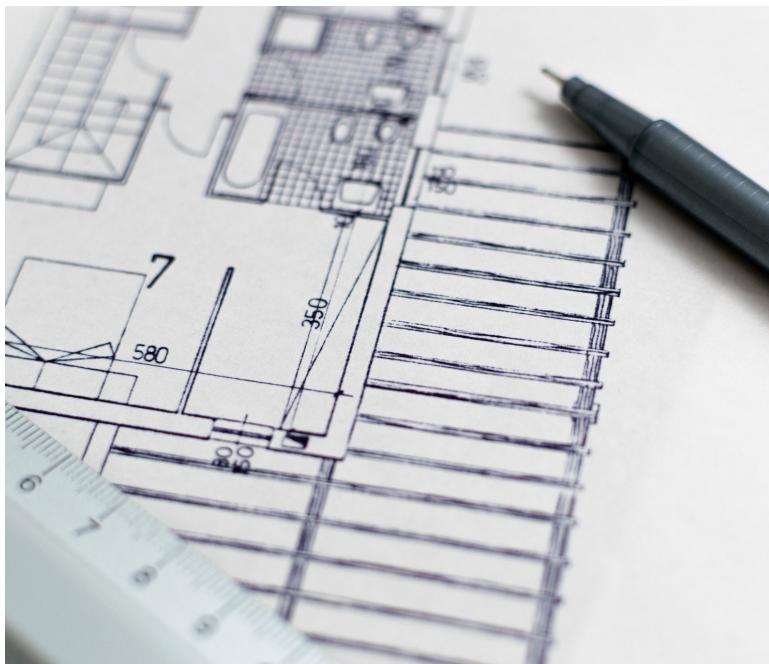


image by lorenzo cafaro from pixabay

**IN THIS SECTION, YOU WILL:** Understand how to use diverse data sources to support architecture decision-making processes and get concrete tips on creating architecture-centric data tools.

**KEY POINTS:**

- The architecture Data Foundation serves as a medium to create a complete, up-to-date picture of critical elements of the technology landscapes of big organizations.
- The Data Foundation provides an architecture-centric view of data about a technology landscape based on source code analyses, public cloud billing reports, vibrancy reports, or incident tickets.
- To facilitate the creation of a Data Foundation, I have been working on creating open-source tools that can help obtain valuable architectural insights from data sources, such as source code repositories.

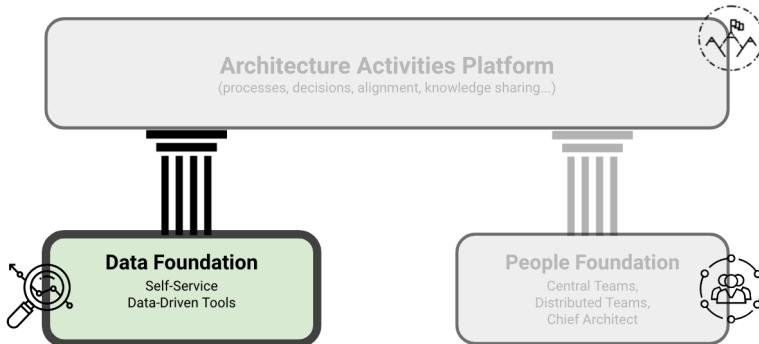
*“If we have data, let’s look at data. If all we have are opinions, let’s go with mine.”* — Jim Barksdale

I strongly emphasized data in every place I worked on creating architectural functions. Moreover, in the past several years, I have been working on creating open-source tools, such as [Sokrates](#)<sup>1</sup>, that can help obtain valuable architectural insights from data sources, such as source code repositories or public cloud billing reports. Consequently, **one of the first steps I make in any architecture practice** is to create an architecture Data Foundation to get a complete, up-to-date picture of critical elements of the technology landscapes of an organization (Figure 1). **Manual documentation does not scale**, and relying on data ensures the reliability and

---

<sup>1</sup><https://sokrates.dev>

scalability of decision-making.



**Figure 1:** The structure of Grounded Architecture: The Data Foundation.

The good news is that **big organizations have lots of data** that, if used wisely, can provide an excellent basis for an architectural Data Foundation. With some automation and curation, getting a good overview of the technology landscape may be closer than it initially appears.

## 5.1: Requirements For A Data Foundation

A Data Foundation should be a central place with **authoritative, relevant, and curated data** about the organizational technology landscape. Technically, you can implement Data Foundation using simple tools like Google Drive, with documents organized in folders or as an internal website. I recommend investing some effort in creating better infrastructure and user experience, as it can enable more people to access and benefit from data.

Simply collecting and putting data in one place will not create any value. Regardless of how you implement your Data Foundation, with papers on the wall, in Google Drive, in Confluence, or with a nicely designed internal website, I have identified the following requirements that a Data Foundation needs to implement:

- **It is the single point of truth** for all relevant architectural data. People should be able to go to one place and get the most relevant data.
- **It is curated for quality** so people can trust the data. Simply dumping data into one place will not help. You need to own curation to ensure that data are correct. You also should provide links to data sources so people can verify the facts.
- **It is curated for usability** so people stay focused on valuable details. You must filter out useless or less relevant details, focusing on the essence. Investing in the UX design of documents or tools you create helps.
- **It is kept up to date**, ideally in an automated fashion (or in a semi-automated repeatable way).
- **It is accessible to the whole organization.** I genuinely believe that when you give employees access to information generally reserved for specialists, architects, or “higher levels,” they get more done independently. They can work

faster without stopping to ask for information and approval. And they make better decisions without needing input from architects or the top.

- **It is used in decision-making.** Having nicely curated and valuable data has zero value if you cannot ensure that such data inform vital decisions.

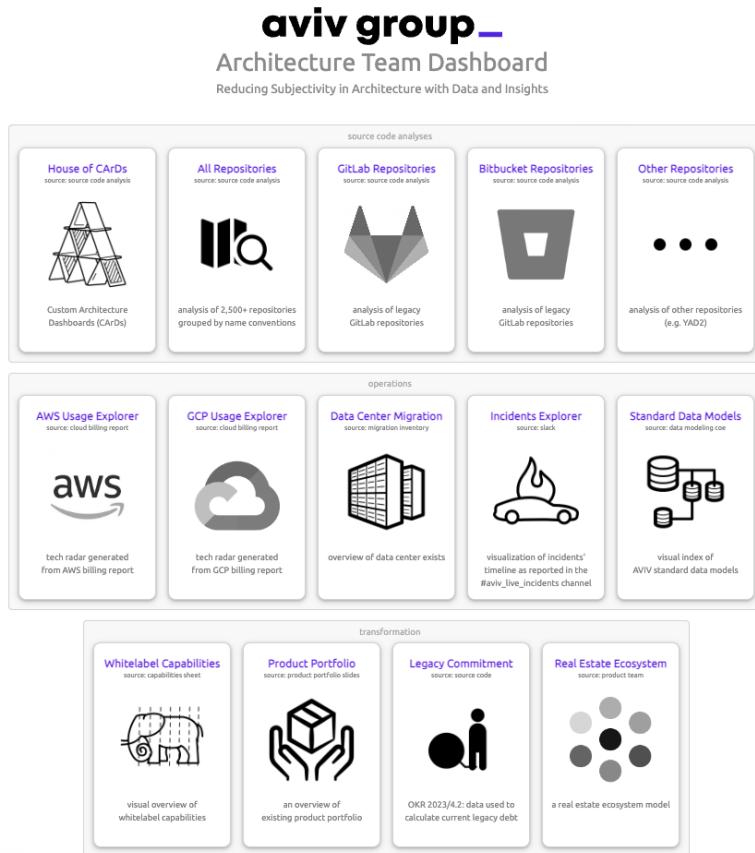
My vision for building the Data Foundation follows **the map-making metaphor**. Maps are one of the most critical documents in human history. They give us tools to store and exchange knowledge about space and place. While there are differences between maps and the layers they show, the one thing that all maps do is **provide readers with orientation**. A sense of place is central to meaning-making. Maps are also composed of multiple layers. Similarly, the architecture Data Foundation should give readers a sense of orientation, offering data layers about systems that describe their sizes, connections, quality, security, or human activity.

## 5.2: Examples of Data Foundation Sources and Tools

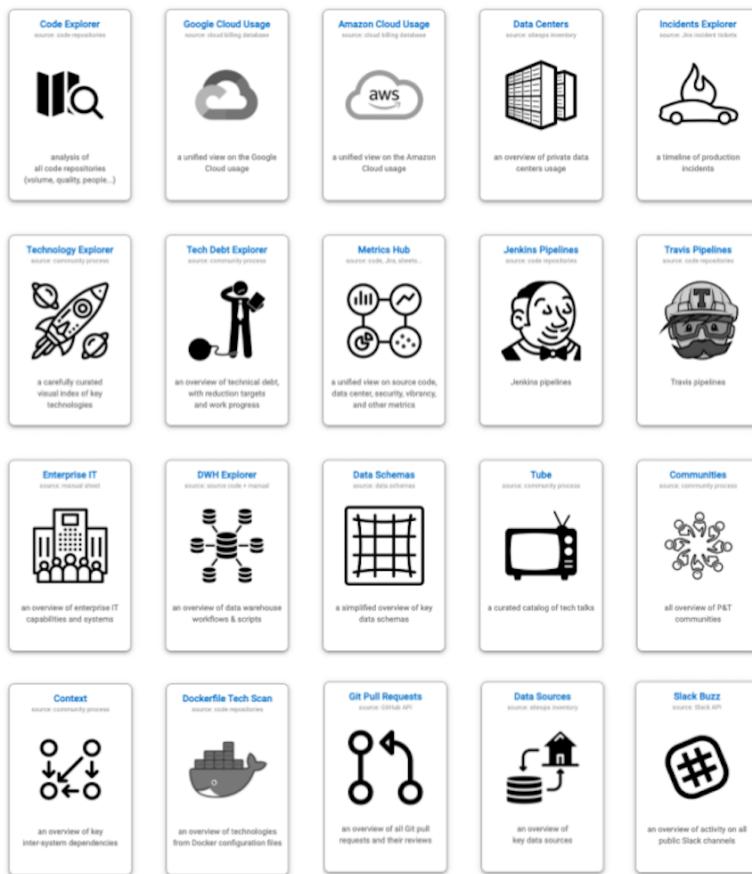
To illustrate what I mean by Data Foundation, I will give a few concrete examples from my recent work. Data I typically used include (Figures 2 and 3):

- **Source code**, which contains an incredible amount of information about technology, people's activity, team dependencies, and the quality of software systems.
- **Public cloud billing reports**, which provide an overview and trends about used cloud services, regions, and budgets.
- **Incident reports**, which can reveal trends and dependencies among incidents.
- **Key business metrics**, like vibrancy, which can show user activity on our systems.
- **Messaging and collaboration platforms (such as Slack) activity reports**, which can help understand discussion topics and team interactions.

In the following sections, I detail several of these architectural data-driven tools.



**Figure 2:** A screenshot of the start page of the architecture data dashboard we've built and used at AVIV Group.



**Figure 3:** A screenshot of the start page of the architecture data dashboard we've built and used at eBay Classifieds.

### 5.2.1: Example 1: Source Code and Commit History

I have repeatably found the source code to be an excellent basis for creating data-driven architecture documentation. Source code and its commit history include an astonishing amount of information

about technology, people activity, team dependencies, and the quality of software systems. As mentioned earlier in this chapter, I have started and still actively maintain the project **Sokrates**<sup>2</sup>, with the idea to help further **extract data from source code that can help my work as an architect**. I use Sokrates daily, improving it on the way.

I have designed Sokrates from an architect's point of view, enabling quick **zooming in and out into source code landscapes**. On the one hand, Sokrates provides a **high-level view** of the IT landscape, summarizing data from all teams and groups. At the same time, you can zoom in on the details of particular systems **to the code level**. That means you can use the same tools to have CTO-level discussions looking at overall trends in technology usage and costs. At the same time, I could engage with developers and discuss concrete code fragments and potential improvements in the code level (e.g., duplicated blocks, complex units, dependencies).

The Appendix at the end of this section shows examples of some insights from source code analyses with Sokrates. For more complex examples of insights that Sokrates generates from source code, take a look at **Sokrates examples**<sup>3</sup>, with analysis of complex open-source landscapes, such as:

- **Apache Software Foundation Repositories**<sup>4</sup>, with aggregated multi-level analysis of more than 1,000 repositories with more than 180 million lines of code, more than 22,000 historical contributors, and 2.4 million commits.
- **Facebook/Meta OSS Repositories**<sup>5</sup>, with aggregated multi-level analysis of around 800 repositories with 120 million lines of code, more than 20,000 historical contributors, and more than 2 million commits.
- **Microsoft OSS Repositories**<sup>6</sup>, with aggregated multi-level

---

<sup>2</sup><https://sokrates.dev>

<sup>3</sup><https://www.sokrates.dev/>

<sup>4</sup>[https://d3axxy9bcycpv7.cloudfront.net/asf/\\_sokrates\\_landscape/index.html](https://d3axxy9bcycpv7.cloudfront.net/asf/_sokrates_landscape/index.html)

<sup>5</sup>[https://d3axxy9bcycpv7.cloudfront.net/meta/\\_sokrates\\_landscape/index.html](https://d3axxy9bcycpv7.cloudfront.net/meta/_sokrates_landscape/index.html)

<sup>6</sup>[https://d3axxy9bcycpv7.cloudfront.net/microsoft/\\_sokrates\\_landscape/index.html](https://d3axxy9bcycpv7.cloudfront.net/microsoft/_sokrates_landscape/index.html)

analysis of more than 2,400 repositories with more than 100 million lines of code, more than 18,000 historical contributors, and more than 1.2 million commits.

- **Google OSS Repositories**<sup>7</sup>, with aggregated multi-level analysis of more than 1,600 repositories with more than 200 million lines of code, more than 27,000 historical contributors, and more than 2.4 million commits.
- **Linux Source Code**<sup>8</sup>, with aggregated multi-level analysis of 178 Linux repository sub-folders with more than 23 million lines of code, more than 17,000 historical contributors, and more than 1.7 million commits.
- **Amazon OSS Repositories**<sup>9</sup>, with aggregated multi-level analysis of more than 2,700 repositories with more than 130 million lines of code, more than 13,000 historical contributors, and more than 600,000 commits.

In addition to standard source code and commit history analyses, I also have built several special source code analyses to get further details:

- Travis and Jenkins analyzers to understand how teams build CI/CD pipelines.
- Dockerfile scan to create a tech radar of runtime technologies.
- GitHub API pull requests analyses to identify deployment frequency.

And I encourage you to experiment with your source-code analyses.

---

<sup>7</sup>[https://d3axxy9bcycpv7.cloudfront.net/google/\\_sokrates\\_landscape/index.html](https://d3axxy9bcycpv7.cloudfront.net/google/_sokrates_landscape/index.html)

<sup>8</sup>[https://d3axxy9bcycpv7.cloudfront.net/asf/\\_sokrates\\_landscape/index.html](https://d3axxy9bcycpv7.cloudfront.net/asf/_sokrates_landscape/index.html)

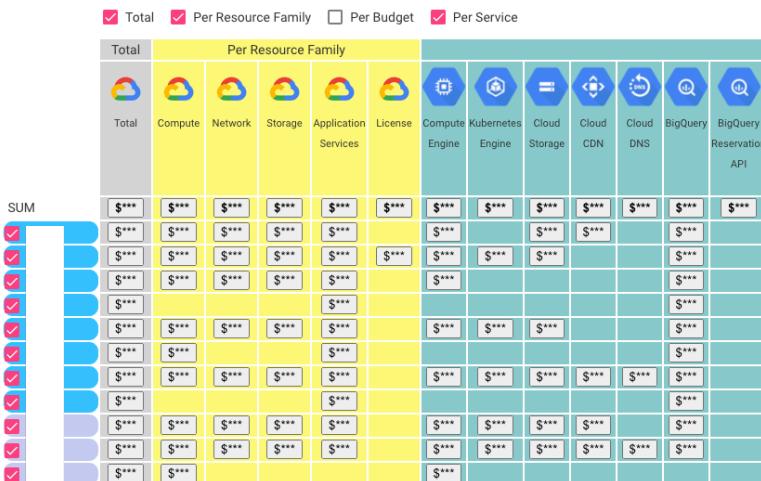
<sup>9</sup>[https://d3axxy9bcycpv7.cloudfront.net/amzn/\\_sokrates\\_landscape/index.html](https://d3axxy9bcycpv7.cloudfront.net/amzn/_sokrates_landscape/index.html)

## 5.2.2: Example 2: Public Cloud Usage

Migrating to the public cloud can dramatically increase transparency thanks to uniform automation and monitoring. The **public cloud transparency** offers incredibly valuable data out-of-the-box.

[Amazon Web Services \(AWS\)](#)<sup>10</sup>, [Google Cloud Platform \(GCP\)](#)<sup>11</sup>, [Microsoft Azure](#)<sup>12</sup>, and other Public Cloud Providers give detailed data about which platform uses which services, resource family, and budget. You can also understand which people and teams have access to each service. Getting real-time information about cloud usage and automatically understanding the trends is straightforward.

Figure 4 shows the anonymous screenshot of the Cloud usage explorer, a tool I built to visualize automatically-collected data from standard Google Cloud Platform (GCP) usage reports.



*Figure 4: An example of a cloud usage explorer.*

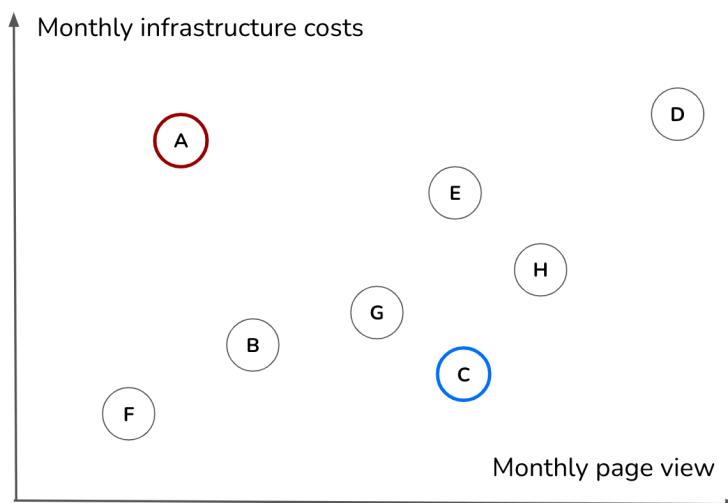
<sup>10</sup><https://aws.amazon.com>

<sup>11</sup><https://cloud.google.com/>

<sup>12</sup>[https://azure.microsoft.com/](https://azure.microsoft.com)

### 5.2.3: Example 3: Financial and Vibrancy Data

Finance departments are very data-driven and have high-quality data that could be relevant for architects. In addition to standard costs, budgets, and other pure financial data types, I frequently found that finance teams also have different data sources, such as **vibrancy or usage levels**. These teams need such data to, for instance, correlate finance performance with usage levels. Such usage data are beneficial for architecture discussions. For example, linking usage levels and vibrancy of systems with their public cloud costs can identify areas of improvement and inefficiencies (Figure 5).



**Figure 5:** Combining data from a different source (e.g., cloud billing reports and vibrancy or revenue can lead to new insights (e.g., identifying inefficiencies in the application portfolio).

## 5.3: Building Data Foundation

While each organization will have its unique sets of data, here are some tips I found helpful in forming the architecture Data Foundation:

- **Start with the source code.** My motto is “*Talk is expensive. Show me the code.*” I scan as soon as possible all source code using tools such as [Sokrates](#)<sup>13</sup>. I highly recommend Sokrates as the basis for the Data Foundation, but other simple analyses could also provide a good starting point. Modern IT enterprises store almost everything as a code. It is the richest and most up-to-date documentation on most things happening in an IT enterprise. I have repeatedly found that people underestimate the size and complexity of organizations and legacy systems. Quick source code scans can reveal such misconceptions, providing a better alternative for long-term tracking of an organizational IT landscape.
- **Connect with finance and governance teams.** My second motto is “*Follow the money!*”. Get exports of finance data (without sensitive parts, such as revenue projections). Cloud billing reports and data about vibrancy or revenue streams are collected anyway. By extracting more technology-oriented data (e.g., public cloud technology usage trends) and connecting them to other data, you can obtain many new insights without starting new processes or asking people to provide more details. First, leverage what you have, squeeze all the value from it, then ask people for any missing elements. In addition, critical exploration of finance data can also benefit finance teams, for instance, by discovering some unused expensive licenses of contracts.
- **Maintain a culture of transparency.** Sharing fewer data with everyone is much simpler and more effective than

---

<sup>13</sup><https://sokrates.dev>

having more data, but you need complex authorization mechanisms.

- **Own the curation.** People need to be able to trust your data. Spend enough time to understand data sets, curate them, and ensure presentation consistency. I consider myself a master curator and chief UX designer of a Data Foundation.
- **Use simple and easy-to-maintain infrastructures.** For example, I publish the results of Sokrates analyses and other data tools as static resources in our enterprise GitHub pages. Configuring more complex infrastructure with complex databases and backend software requires more maintenance. In the [Architecture Dashboard Examples repository](#)<sup>14</sup>, you can find the source code of examples of how to build the Architecture Data Dashboard in such a way. The dashboard is a simple static website generated from JSON files and [published via GitHub pages](#)<sup>15</sup>.

---

<sup>14</sup><https://github.com/zeljkoobrenovic/grounded-architecture-dashboard-examples>

<sup>15</sup><https://zeljkoobrenovic.github.io/grounded-architecture-dashboard-examples/>

## 5.4: Using Architecture Data Foundation

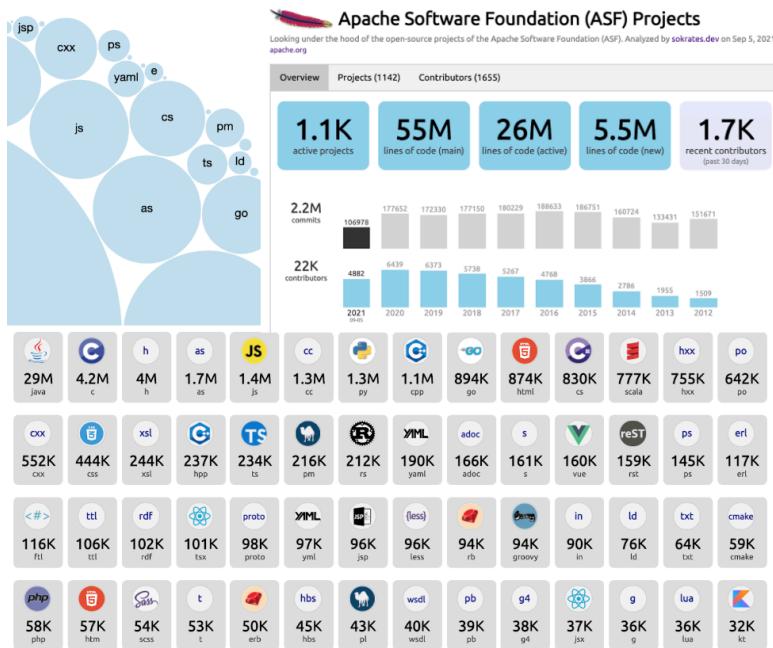
The Data Foundation can provide lots of data. Sometimes, as in an ordinary map or atlas, such data could be helpful for those who want to orient themselves and understand the context. But with a proper mindset, you can obtain more insights from such data.

Finding the right ways to interpret and use data requires active effort. In other words, the data can give you the answers, but you need to bring questions. Here are some of the questions you can ask when you have data:

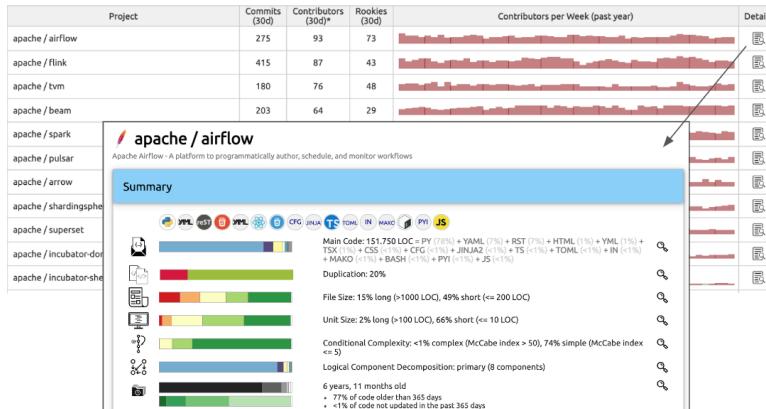
- **Are we aligned and going in the same direction?** Source code overviews, public cloud usage explorers, or tech radars can highlight differences among systems and teams and trigger discussions and actions.
- **Are we using technology optimally?** Comparing usage trends between teams can show interesting outliers (both positive and negative).
- **Are there indicators of poor code quality?** Too large systems, duplication, long units, or long files.
- **Productivity trends: is more really more or is more actually less?** For instance, comparing the number of git merges with the number of developers can indicate if our dev processes are scalable. When we scale up teams, we want to speed up our delivery (but if team structure is not proper, it can easily be the opposite as people “step on each other toes”).
- **Do we collaborate in the way we want?** Repository analysis can point out team topologies and (un)desired dependencies.
- **Do we work on the things we want?** We may want to focus more on innovations, but in reality, we may spend too much time on legacy maintenance.

## 5.5: Appendix: Examples of Insights From Source Code Analyses

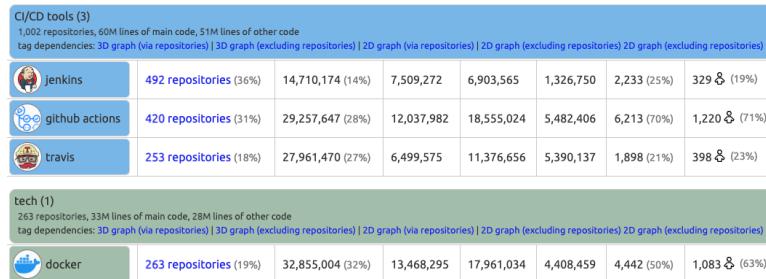
Figures 6 to 10 show some insights from source code analyses with Sokrates.



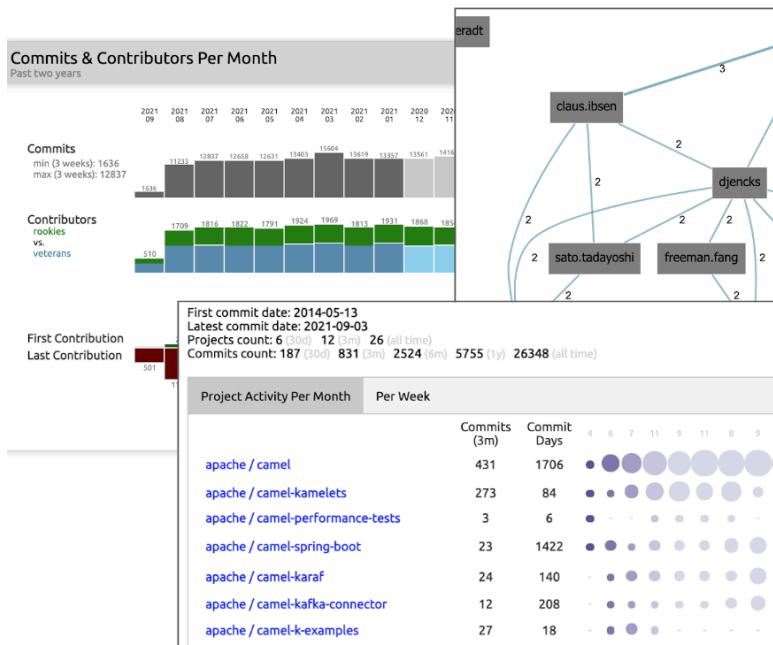
**Figure 6:** Sokrates can instantly create a helicopter view of the technology landscape, programming languages, active contributors, and commit trends.



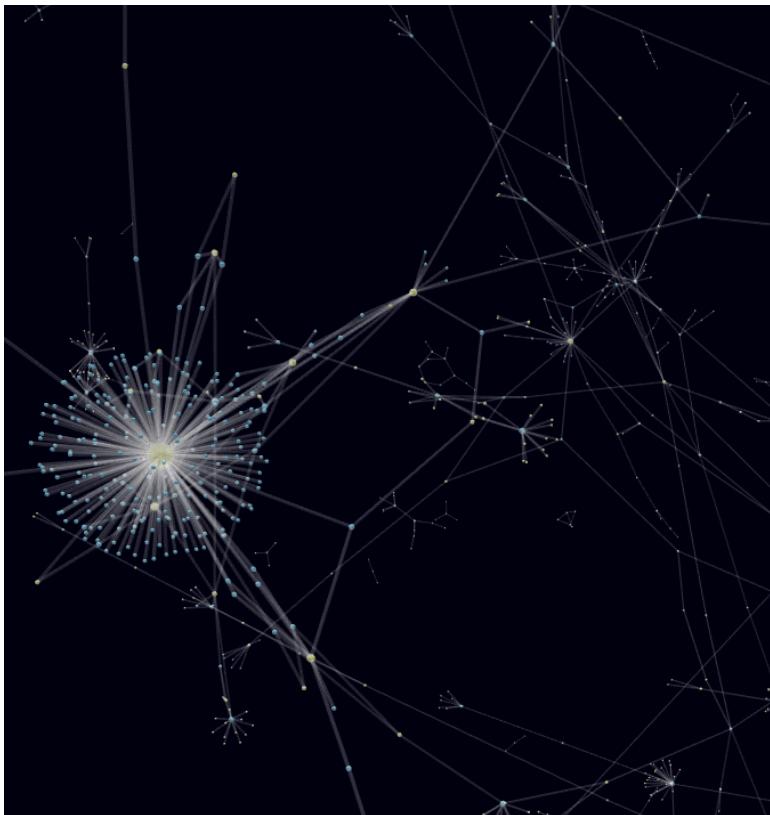
**Figure 7:** Sokrates can show detailed code and contributors' trends per repository, enabling zooming in each repository up to the code level.



**Figure 8:** Sokrates can create a tech radar by tagging projects with identified technologies.



**Figure 9:** Sokrates can show contributor trends, distribution of “veterans” and “rookies,” and dependencies between people and repositories, enabling zooming in into patterns of the contribution of individual contributors.



**Figure 10:** Sokrates can reveal the team topologies by plotting 2D and 3D graphs of dependencies that people create through working on the same repositories in the same period.

## 5.6: Questions to Consider

Using data can significantly improve the efficiency and impact of architectural practice. But there are no simple tools that can instantly provide you insights. Ask yourself the following questions:

- *Have you considered using open-source tools like Sokrates to gain architectural insights from data sources? Why or why not?*
- *What are your views on the reliability and scalability of manual documentation?*
- *What steps would you take to create an architecture Data Foundation in your organization?*
- *Are there untapped data sources within your organization that could inform your architectural decisions?*
- *How could you automate gathering data for architectural insights in your organization?*
- *What examples can you provide of the data you've used to gain reliable information about technology in your organization?*
- *How would you examine public cloud billing reports, incident reports, or key business metrics for architectural insights?*
- *How can you ensure your data is reliable and up-to-date?*
- *Do you collaborate with finance and governance teams to incorporate financial and vibrancy data into your data analysis?*
- *Is there a culture of transparency in your organization?*

## 6: People Foundation



image by mostafa meraji from pixabay

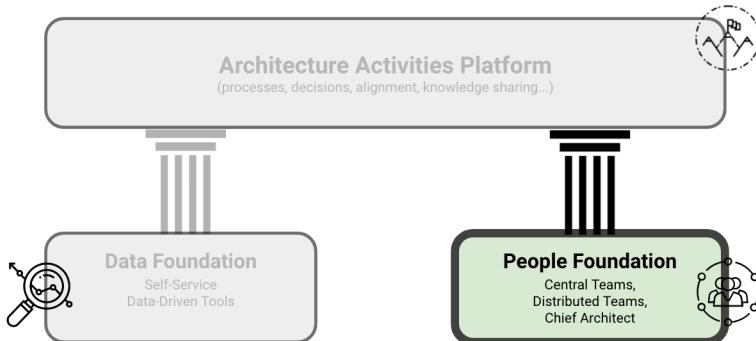
**IN THIS SECTION, YOU WILL:** Understand that architecture practice is all about people and get tips on creating organizational structures that support practical IT architecture practice.

**KEY POINTS:**

- Developing the architecture function requires having competent, empowered, and motivated architects. Architecture practice must carefully organize, empower, and leverage scarce talent.
- In my work in the past few years, I combined two teams of architects: a small central architecture team and a cross-organizational distributed virtual team.

The **People Foundation** is an essential element of Grounded Architecture. As noted by Gregor Hohpe, to transform an organization, you do not need to solve mathematical equations. You need to move people. Consequently, having a **strong network of people doing architecture** across the organization is crucial to ensure that the architecture function has any tangible impact. In other words, **Strong Architecture = Strong Architects**.

Developing the architecture function requires having competent, empowered, and motivated architects. **We should not take architectural talent for granted.** Architects, and other people doing architecture work, are bridging local business, product, organizational, and technology issues. Architects are difficult to hire talent as they need not only in-depth technical knowledge but also domain-specific and organizational knowledge. Consequently, any architecture function must carefully organize, empower, and leverage scarce architecture talent (Figure 1).

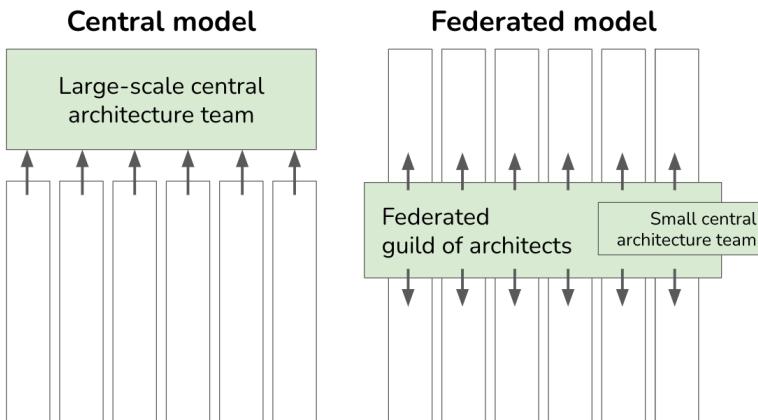


**Figure 1:** The structure of Grounded Architecture: The People Foundation.

In my work in the past few years, I was working by combining, in different forms, two teams of architects: a small **central architecture team** and a **cross-organizational distributed virtual team**. A central architecture team is an enabler for the rest of the organization, supporting other teams and addressing global strategic topics. A distributed virtual architecture team consists of architects (or other people making architecture decisions in their teams) working in local organizational units but spending some time in a virtual team with peers from other teams. Such a distributed virtual architecture team is a crucial element of an architecture function. It provides the connection (grounding) across all parts and levels of the organization, increasing transparency, building people networks, and making it easier to implement change.

## 6.1: Background: Central vs. Federated Architecture Function

IT architecture practices generally follow one of two fundamental models: central or federated (Figure 2, [McKinsey 2022<sup>1</sup>](#)).



*Figure 2: Central vs. Federated Architecture Function.*

**The central model** involves a large-scale central architecture team. The central team typically defines the process for approval of new work and assures adherence by development teams. In this model, development teams have few or no qualified solutions architects. This model also holds **centralized infrastructure, operations, and security teams** apart from the development function. **Control and governance** are typically the primary concerns of the central architecture team.

**The federated model** generally relies upon a guild or “community of practice” of solutions architects embedded into individual development teams. A small central architecture team or an **architecture center of excellence (CoE)** may complement such a guild.

---

<sup>1</sup><https://www.mckinsey.com/business-functions/mckinsey-digital/our-insights/tech-forward/crafting-the-optimal-model-for-the-it-architecture-organization>

The federated model's architects facilitate high-level planning and act as **on-demand service providers** for distributed teams.

The **federated model** is more commonly associated with **cross-functional DevOps culture**. The roles of solution and enterprise architects are generally broader in scope to integrate infrastructure, operations, and security concerns in product-oriented teams. The architect's role focuses on **facilitation and enablement** rather than control.

Today, modern agile organizations mainly adopt the federated model. This approach increases the likelihood that the central architecture team will spend time closely involved with the challenges identified in the teams. The model ensures that the architects will be evaluated against the goals of the individual products they support, thereby focusing on improving performance and reducing complexity.

## 6.2: The Hybrid Model

To operate in a complex [context](#), you must invest effort to ensure you have the right people at the right places. In the end, I usually found it best to adopt a model of a hybrid organization combining elements of central and federated orientation structures:

- A Central Architectural Teams, and
- Architecture Guilds & Virtual Architectural Teams.

This model is similar to the previously described **federated model** but with extra investment in a central team that should be more than just an on-demand service provider.

The **hybrid team structure** executing at scale in complex organizations:

- **Guilds** and virtual architecture teams support execution by **increasing the number of people involved** in architectural activities and increasing work efficiency through better alignment.
- Members representing various organizational units can have much **more impact across the board**.
- And having some capacity on the **central level** serves as a **catalyst helping people at local levels** to do their job while being aligned and better connected with overall strategic goals and other teams working on related topics.

### 6.2.1: Central Architecture Team

The roles of people in central teams may differ depending on the organization. In addition to doing typical architecture work, I found it helpful to be able to have dedicated people that can cover the following types of responsibilities:

- **Build and maintain the architecture Data Foundation.** Building a Data Foundation will not happen by accident. It requires clear ownership, curation, and technical support.
- **Promote data-informed decision-making.** Identify, collect, and use relevant data. Only some people are used to applying data in their decision-making. Architects should provide support and be a role-model for data-informed organizations.
- **Proactively identify, connect, and maintain relationships with all relevant stakeholders.** Architects are frequently uniquely positioned to bridge different organizational units and stakeholders.
- **Build internal architecture communities and guilds.** Organizing rituals and people requires active effort.

While guilds and virtual teams could do many listed activities, the voluntary and typically less formal nature of guilds and virtual groups makes such support more fragile. The central architecture team can take full long-term ownership of some topics and be a backup if community support weakens, ensuring long-term continuity.

### 6.2.2: Architecture Guilds & Virtual Architecture Teams

*“A lot of cheap seats in the arena are filled with people who never venture onto the floor. They just hurl mean-spirited criticisms and put-downs from a safe distance. ... we need to be selective about the feedback we let into our lives. For me, if you’re not in the arena getting your ass kicked, I’m not interested in your feedback.” — Brené Brown, Rising Strong*

I always found it essential to connect organization members passionate about architecture in some form, a guild, a community of interest, or a virtual team.

Guild or virtual teams work most of the time as architects or tech leads in specific organizational units but spend **part-time collaborating with architects or tech leads from other departments** to reach more alignment, share knowledge, and leverage each other's work. In this peer-to-peer community, architects are collectively responsible for identifying and growing architectural talent, mentoring, and helping each other.

When having many guilds and teams, we typically have organized architects in several sub-areas:

- **General or core teams** for a broader set of architectural topics.
- **Specialist teams** focus on the architectures of a particular part of the technology stack. Examples include native mobile apps, web frontends, public cloud, etc.
- **Strategic initiatives teams.** For instance, data strategy, public cloud strategy, transactions, or verticalization.

Having places and events to connect central and distributed teams is essential. Such events can transform individual experiences into collective knowledge that can benefit the whole organization. In most organizations I worked in, such events followed a similar pattern of rituals:

- **Regular (e.g., bi-weekly) forums**, with updates, announcements of architectural spikes, and sharing or architectural decisions (similar to Andrew Harmel-Law's [Advice Process](#)<sup>2</sup>)
- **Annual or bi-annual summit** or architecture days, with several days of intensive knowledge sharing and workshops

---

<sup>2</sup><https://martinfowler.com/articles/scaling-architecture-conversationally.html>

- **Ad-hoc workshops** focusing on some specific topic.

While the central team can provide some essential support, all communities must take the initiative and engage as many people as possible during these events. People should be active participants rather than passive receivers of information to ensure more involvement and commitment.

### 6.2.3: Embracing Diverse Team Structures

When building architecture guilds and virtual architecture teams, it is essential to acknowledge that **organizational units have diverse structures and sizes**. In big organizations, embracing diversity is a pre-requirements to have any broad impact.

There is no one-fit-all solution about how departments should assign architecture responsibilities. I generally worked on three types of team-architect systems per [Gregor Hohpe's view of architects and their teams' relationships<sup>3</sup>](#):

1. **Benevolent “dictator”**: An architect or architect team tells developers what to do or how to do things. An important nuance is to what extent the line is unidirectional or bi-directional.
2. **Primus inter pares (first among equals)**: Architects are embedded into teams where each is just another team member focusing on the system structure and trade-offs, taking a longer-term view than other team members.
3. **Architecture without architects**: Architecture is done within teams. However, the task is a shared responsibility across multiple (or all) team members. This approach is often the preferred model in modern technology organizations.

---

<sup>3</sup><https://architectelevator.com/architecture/organizing-architecture/>

## 6.3: Building People Foundation

While each organization will need its unique approach, here are some tips I found helpful when forming architecture teams and building a “[People Foundation](#)”:

- Before making grandiose plans for reorganizations, **connect with the people already doing architectural work** in an organization, creating a community of practices or a virtual team. Being inclusive and connecting all key tech leaders, regardless of their actual position and title, is vital. Being well-connected to these people will be crucial in any architecture organization, so you will always benefit from this effort.
- If creating a virtual team is a part of your architecture strategy, move away from making an informal community of practice towards **building a team with more accountability and responsibility**.
- **Connect with non-architecture stakeholders** early to gain broader support for building architecture teams and guilds. Again, being well-connected to these stakeholders will be crucial in any architecture organization.
- **Avoid hiring a digital hitman**<sup>4</sup>. Invest in growing internal talent. Architects need knowledge of technology, domain, and organization. Finding such a person outside the organization is challenging.
- **Externalize**. Reach out and connect. Participate in external events. Publish. Being strong externally can help you to both grow and attract architectural talent.

---

<sup>4</sup><https://architectelevator.com/transformation/dont-hire-hitman/>



image by chantellev from pixabay

## 6.4: Questions to Consider

It is difficult to overestimate the importance of people for architecture practice, yet many organizations take architectural talent for granted. To reflect on the importance of carefully organizing, empowering, and leveraging scarce architecture talent, ask yourself the following questions:

- *Reflect on your organization's current architecture function. Do you have a strong network of architects across the organization?*
- *How do you ensure architects' competency, empowerment, and motivation in your organization? What systems do you have in place to develop architectural talent?*
- *Which central, federated, or hybrid model best represents your current architectural function? Why was this model chosen, and how effective has it been for your organization?*
- *If you are part of a central architecture team, how would you support the rest of the organization? How would you contribute to the global architecture function if you were part of a distributed virtual team?*
- *Consider having the roles of central architecture teams and federated architecture teams in your organization. How would they complement each other?*
- *How effective is the current division of responsibilities among architects in your organization? Are there areas of overlap or gaps in coverage?*
- *What steps has your organization taken to ensure architects are well-connected across all parts and levels? What impact has this had on transparency and the implementation of changes?*
- *Reflect on the diversity of team structures within your organization. How does this diversity impact the roles and responsibilities of architects?*

# 7: Architecture Activities Platform

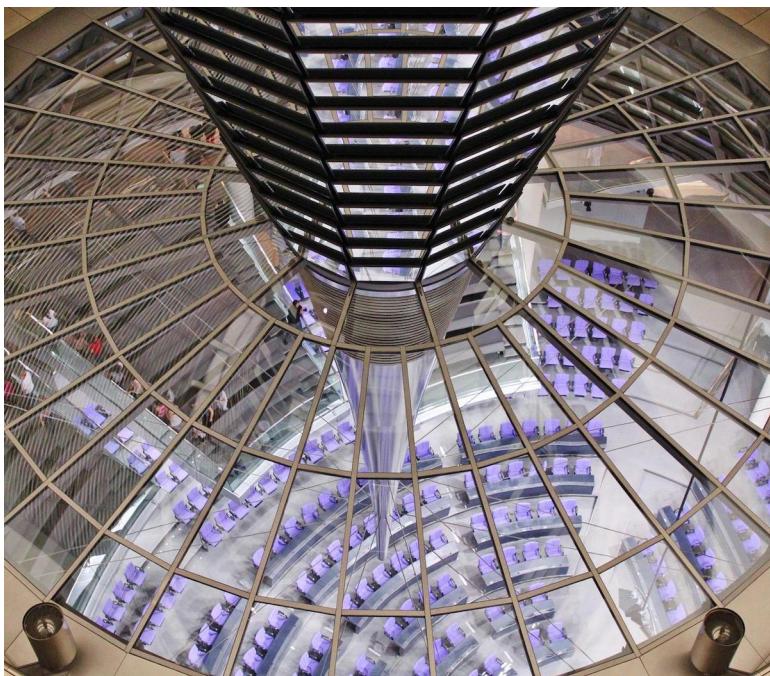


image by anja from pixabay

**IN THIS SECTION, YOU WILL:** Understand what activities you can do as a part of architecture practice and get tips on creating pragmatic operating models for an architecture practice.

**KEY POINTS:**

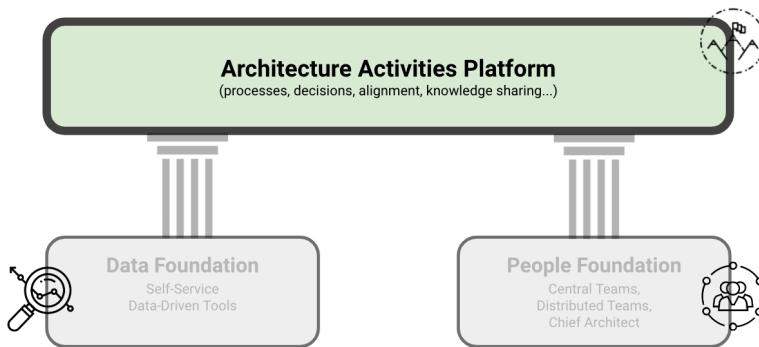
- The Architecture Activities Platform is a system of processes and agreements enabling architects to do everything architecture typically does, leveraging Data and People Foundations to create a data-informed, organization-wide impact.
- Examples of activities include: supporting teams in their daily work; tracking tech debt, defining tech debt reduction programs; performing technical due diligence; standardizing processes and documentation; defining cloud, data, and platform strategies.

Each organization will have different architectural needs and contexts. When forming architecture functions, I use as a starting point these **two pieces of advice from Gregor Hohpe<sup>1</sup>**:

- “*Your architecture team’s job is to solve your biggest problems. The best setup is the one that allows it to accomplish that.*”
- “*Your organization has to earn its way to an effective architecture function. You can’t just plug some architects into the current mess and expect it to solve all your problems.*”

---

<sup>1</sup><https://architectelevator.com/architecture/organizing-architecture/>



**Figure 1:** The structure of Grounded Architecture: Architecture Activities Platform.

Considering the previous two points from Gregor Hohpe, I approach defining an architecture practice with a mindset that there is no one-size-fits-all approach. You must find your own activities and operating models to enable architecture to solve the most critical problems.

No matter which operating models you select, you must develop explicit agreements and “rules of engagement” with key stakeholders to create a **sustainable and practical architecture function**.

The Architecture Activities Platform (Figure 1) is a set of **processes and agreements** that allows architects to do everything architecture practice typically does, leveraging Data and People Foundations to develop a data-informed, organization-wide impact. In all these activities, Data and People Foundations provide a basis for data-informed decision-making well embedded in the organization.

## 7.1: Examples of Architecture Activities

To better understand what I mean by an Architecture Activities Platform, here are some examples of activities I have been performing with architects:

- **Designing mechanisms for teams to make better decisions.** This activity includes creating global decision-support mechanisms, such as [advisory forums](#)<sup>2</sup>, formal design authority (for compliance-sensitive projects), and team-specific mechanisms, such as escalation paths in case of decision conflicts (e.g., teams cannot align on a common messaging middleware).
- **Supporting teams in their daily work.** Being part of key team activities, aligning architectural work with team rituals to provide timely support, and supporting the team in all crucial phases of their work (e.g., reviewing architecture proposals early before the project or sprint starts).
- **Supporting planned new initiatives and projects.** Ensuring alignment between projects that require multi-team collaboration.
- **Supporting teams in dealing with the legacy landscape.** Providing data and knowledge regarding legacy landscape, identifying hotspots (e.g., frequently changed, low-quality untested pieces of legacy code), defining scenarios and roadmap for legacy modernization.
- **Tracking tech debt, defining tech debt reduction programs.** Defining a centrally aligned backlog of technology depth, defining programs for its reduction, and integration in planning processes.
- **Performing SWOT<sup>3</sup> and other analyses of platforms and systems.** Doing deep dives to better understand some areas

---

<sup>2</sup><https://martinfowler.com/articles/scaling-architecture-conversationally.html>

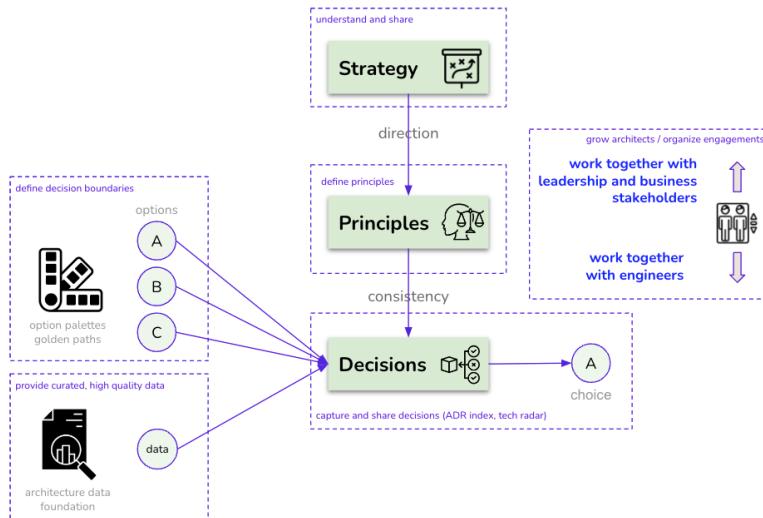
<sup>3</sup>[https://en.wikipedia.org/wiki/SWOT\\_analysis](https://en.wikipedia.org/wiki/SWOT_analysis)

of the technology landscape and create plans and roadmaps for improvement.

- **Standardizing of processes and documentation.** Defining standard templates for documents such as Architectural Decision Records (ADRs), Technical Design Reviews (TDRs), or common diagrams.
- **Supporting merger and acquisitions (M&A) activities with expertise and analyses.** Support analyses, recommendations, and integration planning regarding mergers and acquisitions.
- **Defining key technology strategies.** Examples include Cloud, Data, and Platform strategies.
- **Defining vision and direction of technology, frequently collaborating with Engineering Leaders.** Working with managers to create a sustainable organizational setting aligned with technology strategies.

## 7.2: Operating Model

While exact activities and their scope will depend on an organization setting and will change over time, I usually followed a common operational model in daily work (Figure 2).



**Figure 2:** A common operating model I typically use for Grounded Architecture activities.

Inspired by Gregor Hohpe's strategy-principles-decisions model, I typically used these guidelines:

- **Engagement mindset:**
  - Architects engage with stakeholders and product and project teams in a **collaborative and supportive manner**.
  - Architects aim to **empower the teams** so that they make most of the decisions.
- **Role of architects:**

- Bring relevant data to inform decisions leveraging a [Data Foundation](#).
- Define decision boundaries to enable minimal compatibility and strategic alignment (e.g., public cloud provider, golden paths, or tech stack constraints).
- Define fundamental principles to facilitate consistency in decision-making.
- Share and generalize lessons learned via a [People Foundation](#).

- Position of architects:

- Architects spend their time in **constant motion** between supporting teams' **daily work** and working on **strategic topics**, helping the organization achieve alignment between strategy and implementation.

Another characteristic of this operating model is **shifting left** the architecture work. I aim to avoid formal bureaucratic approval processes, where architects appear too late and are frequently busy approving trivial decisions. Instead, my goal is to have architects involved early in any of the processes, such as during the planning and preparation stages, where it is possible to make more significant changes.

## 7.3: Architecture Decision Policy

Inspired by the famous [Netflix expense policy](#)<sup>4</sup>, I frequently argued that architecture decision policy could be summarized in six words:

“Decide in the Organization’s Best Interests.”

What I mean by that is that anyone can make architecture decisions, provided that, in addition to their specific requirements, they also think about the impact of their choices on:

- **Overall organizational complexity:** Technology is more manageable by limiting tech diversity, size, and dependencies. Limiting technology choices also reduces the attack surface with fewer third-party dependencies and tool ecosystems (build, testing, etc.).
- **Ease of moving people between teams** (both to get help and help others, get promoted): Do not unnecessarily create exotic islands with few experts in technologies not supported or widely used in the organization. People cannot get help or move across the organization as their expertise may be useless outside the team.
- **Ease of training and onboarding** of internal and external developers: Using conventional technologies, supported with external learning resources (books, tutorials, StackOverflow) significantly helps find and grow experts.

---

<sup>4</sup><https://hbr.org/2014/01/how-netflix-reinvented-hr>

- **Talent density** and the possibility of performing at the world-scale level: Building world-scale technology and scaling requires in-depth knowledge and fine-tuning. You cannot achieve it with only a few in-house experts.
- New **reorganizations**: Would your choices fit with other components from other areas if ownership of components changes (e.g., another team is taking it over)?
- Reducing global **duplication of effort** and inefficiencies: Are you doing the work others are doing? Can others reuse your work? Can you reuse the work of others?

While it may not always be enough, this simple policy resonates well and encourages people to be more thoughtful when making decisions.

## 7.4: Distributing Decisions, Autonomy, and Alignment

With any operating model, I aim to keep architectural decision-making distributed across the organization and embedded in the development teams. **Development teams traditionally have the best insights and most information** relevant for making decisions. As noted by Gregor Hohpe, the worst case of organizational decision-making happens when people with relevant information are not allowed to make decisions, while people who lack sufficient information make all decisions. Grounded Architecture aims to make relevant information more readily available to a broader audience and better connect people when making decisions.

While I aim to create a mechanism to give teams autonomy, autonomy does not mean that teams are alone and do not align with anyone, do not get feedback from anyone, and do whatever they want. Teams must complement autonomy with high transparency and proactivity in alignment with other groups.

To give maximal autonomy to the teams while maintaining a **minimal level of global alignment and compatibility**, I have sometimes implemented the concept of a **decision pyramid** (Figure 3).

The **decision pyramid** highlights that development teams should make most decisions. However, several strategic and area-level choices may provide team decision boundaries. For example, selecting the public cloud provider is typically a CTO-level strategic decision. Similarly, engineering leaders in some areas may want to limit some choices, such as the number of programming languages, to more easily train new people, maintain code, and support moves between teams.



*Figure 3: A decision pyramid. The development teams should make most decisions. However, several strategic and area-level decisions may provide decision boundaries for teams (e.g., a public cloud provider).*

## 7.5: Rules of Engagement

One of the amusing challenges with setting up an architecture function in an organization is that everyone seems to have a different idea of what “architecture” should entail. It’s like asking people to describe a unicorn: some imagine a mythical, majestic creature, while others picture a sparkly horse with a horn that grants wishes. Good architects can do many things, but this versatility might not always be the most effective way to support the organization.



image by gordon johnson from pixabay

To be effective, I've found it crucial to establish and clearly communicate some “rules of engagement” (ROE). Think of ROE as the office playbook for how architects should operate. In a corporate setting, ROE are the principles that guide how employees and departments interact with each other, clients, and stakeholders. This includes communication protocols, decision-making processes, and conflict-resolution mechanisms. Essentially, ROE sets the stage for

what's expected and what's not, ensuring everyone plays nicely and fairly.

While you may need to tailor these rules to fit your organization, I found it helpful to set expectations for what the team should be able to do to qualify for the architecture support. Here's a handy list of expectations for teams seeking architecture support. This also helps clarify what architecture practice isn't supposed to do:

1. **Organizational Awareness and Connections:** Teams should know all relevant stakeholders and actively engage with them. This includes product, development, and business stakeholders. Planning should be collaborative across all affected teams, with active working relationships with global functions like QA, DevOps, or Security.
2. **Enough Capacity and Skills:** Teams should have adequate development capacity with the right skills and seniority to innovate and maintain their products.
3. **Strategic Awareness:** Teams should understand the organization's strategic goals, technologies, and other relevant strategies, and know their role within these frameworks.
4. **Technical Documentation Literacy:** Teams should be capable of creating technical documentation, such as ADRs (Architecture Decision Records) or RFCs (Request for Comments).
5. **Technology Standard Awareness:** Teams should be familiar with the organization's technology standards, including golden paths and guidelines for planning, documentation, security, DevOps, and QA processes.
6. **Participation and Citizenship:** Teams should actively participate in relevant communities (like architecture guilds) and global events (such as architecture summits).
7. **Tech Debt Management:** Teams must be aware of the technical debt they create and maintain, ideally having a tech debt backlog and a plan for "paying" it back.

Aligning on these rules with the teams helps ensure productive conversations about architectural support. When these conditions are met, architecture practice can help teams level up. When they're not, architecture support can't be as effective. However, that doesn't mean struggling teams are left in the lurch. Architecture can help teams meet these expectations but can't compensate for their total lack. Teams need to take the initiative and lead. For instance, it's impractical to have architects working full-time for months with one team as their senior developer. However, architects can coach and help developers grow. Similarly, architects can assist in building relationships with other teams, but the teams themselves need to be active and engaged.

So, set those expectations, establish your rules of engagement, and watch as your architecture function goes from a sparkly unicorn to a well-oiled machine!

## 7.6: Questions to Consider

Your architecture practice job is to solve the biggest problems in your organization. Ask yourself the following questions:

- *How can you identify the most critical problems that your architects need to solve in your organization?*
- *What activities and operating models can you think of that will best enable architecture in your organization to work on these critical problems?*
- *What does the Architecture Activities Platform look like in your organization, and how could it be improved?*
- *Which of the provided examples of architectural activities are you currently performing in your organization?*
- *How does the proposed common operating model align with your current operational practices in your organization? What changes might be necessary to adopt this model?*
- *In your experience, how early are architects involved in projects and activities? Do you agree with the goal of ‘shifting left’ the architecture work?*
- *How are architectural decisions distributed across your organization currently? How could this process be improved to ensure the people with the most relevant information make the decisions?*
- *Reflect on the balance of autonomy and alignment in your organization. How could you better implement a mechanism to give teams autonomy while maintaining alignment and compatibility with global strategy?*
- *How does the concept of a decision pyramid resonate with you? How is it reflected in your current organization, and how could it be better implemented?*
- *Which strategic and area-level decisions provide team decision boundaries in your organization? Are there areas where you need more or less limitations to optimize performance?*

# **8: Value of Grounded Architecture Structure**



image by matthias wewering from pixabay

**IN THIS SECTION, YOU WILL:** Understand the value that architecture practice based on the ideas of Grounded Architecture can create for an organization.

**KEY POINTS:**

- When a Grounded Architecture structure is in place, it can significantly impact an organization's functioning.
- These impact categories are Executing At Scale, Improving the Quality of Decision-Making with Data, Maximizing Organizational Alignment & Learning, and Higher Adaptivity.

When a Grounded Architecture structure is in place, it can have a significant positive impact on the functioning of an organization. These categories of impact, aligned with [defined goals](#), are:

- Enabling Execution of Architecture Function At Scale,
- Increasing Architecture Function Adaptivity,
- Improving the Quality of Decision-Making with Data,
- Maximizing Organizational Alignment,
- Maximizing Organizational Learning, and

## 8.1: Executing at Scale

The first goal was to find a way to support hundreds of teams, and thousands of projects, with significant complexity and diversity.

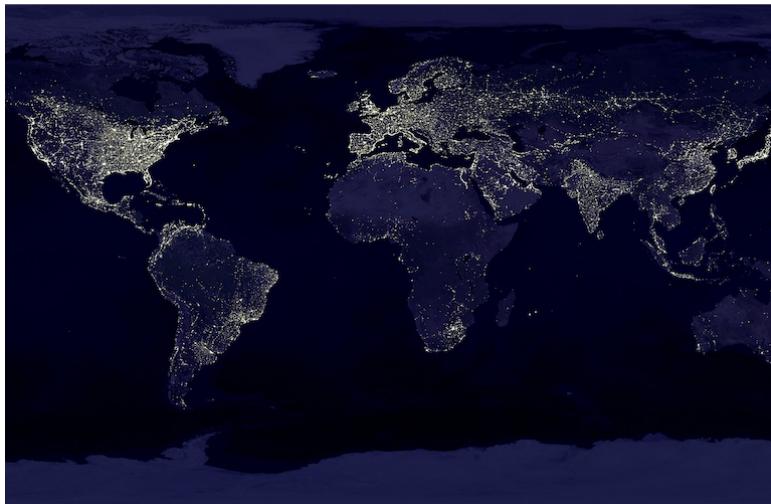


image by wikiimages from pixabay

Grounded Architecture structure can enable architecture functions to operate at scale in the following ways.

- The **Data Foundation** can support working at scale when implemented with a high level of **automation** (as manual effort does not scale) and when data are provided to the organization as a **self-service** (so that we do not rely on manual work or meetings to share the data). Data Foundation tools and insights it displays can reach thousands of people in an organization, e.g., via an internal website. Having such tools has removed the need for hundreds of information-sharing or data-gathering meetings and workshops in my work (that do not scale).

- The **People Foundation** can help execution at scale by developing connections at all levels of the organization, **speeding up alignment, information sharing, and the execution of shared decisions.**
- The **Architecture Activities Platform** enables execution at scale by promoting an operating model that **distributes decision-making** across the organization. In that way, more people can make decisions, removing a bottleneck that centralized decision-making would create.

## 8.2: Adaptivity

The second goal states that architectural functions must adapt quickly to stay relevant in new contexts.



image by francis ray from pixabay

The three elements of the Grounded Architecture structure provide a highly flexible and adaptive setting. This adaptivity is driven by the independence of these elements and the possibility of using the elements in different combinations. Here are some of the critical drivers of flexibility:

- The **Data Foundation**, if implemented with a high level of automation, allows for **quick extensions and reconfigurations** to provide data for any organizational change. For instance, extending the platform, e.g., with additional analyses of new source code repositories and new data after acquisitions or mergers. A robust, automated Data Foundation provides crucial connections and feedback, adapting its views to the changing daily reality of each part of the organization.

- Having the **People Foundation** enables a more flexible and sustainable architectural function. A central team can help mitigate distributed teams' temporary lack of capacity. And with well-connected architects, the architecture function can still benefit the organization without a strong central team.
- The **Architecture Activities Platform** can support adaptivity through its **flexible setting** and distribution of decision-making across the company. This way, we can avoid an architecture function becoming a simple point of failure and a bottleneck.

Grounding the architecture with data and people connections also makes the work of **most senior architects much more flexible**. As they can delegate most architectural decisions to teams, the most experienced technologists can be available to spend more time on crucial strategic initiatives, such as defining cloud, data, or platform strategies or supporting decisions on mergers and acquisitions.

## 8.3: Improving the Quality of Decision-Making with Data

The third goal stated that we need tools and mechanisms to make a decision process more data-informed and less dependent on opinions. There are significant benefits to making our decision process as much as possible data-driven. Architectural **discussions can be heated and opinionated**, not leading to the best arguments and decisions.

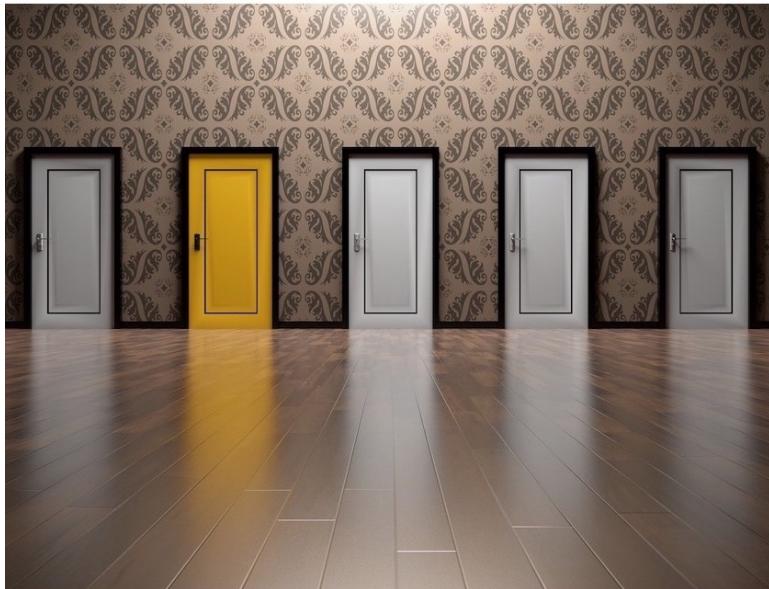


image by arek socha from pixabay

Maintaining high-quality data on relevant internal and external technology developments is one of the critical tasks for architects.

- The **Data Foundation** can ensure having ready any data needed for decisions, fueling data-informed discussions and

decision-making.

- The **People Foundation** ensures having the right people available and well-connected for sharing information and making decisions.
- The **Architecture Activities Platform** provides processes enabling architects to move from opinion-based decisions to data-driven economic risk modeling. Such processes can help architects to achieve the following:
  - **dismantle hype and buzzwords**, present the problem in clear terms, understandable to a broader audience
  - **identify the critical drivers** behind hype and buzzwords based on internal and external research
  - **bring data** into the discussion
  - **translate drivers and data into economic risk models**, and use the models and data to find the best spot for the given business context

## 8.4: Maximizing Organizational Alignment

The fourth goal stated that the architecture function should be a cohesive factor in minimizing misalignments.

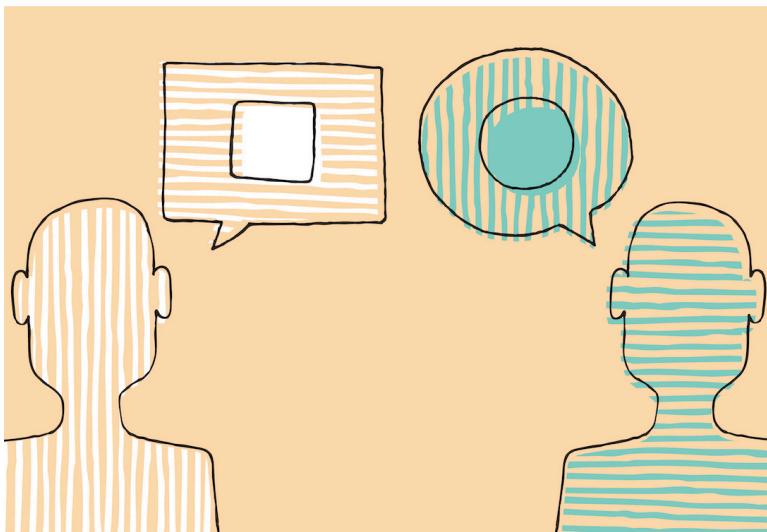


image by istock

Misalignment frequently happens in big organizations. Grounded Architecture structures can minimize such misalignments via the following:

- The **Data Foundation** can increase organizational alignment by creating transparency.
- The **People Foundation** develops global structures to connect people and make it easier for them to collaborate.
- The **Architecture Activities Platform** provides processes helping people to increase alignment, both before and after making decisions:

- **Before making a decision**, people starting to work simultaneously on the same topics can decide to work together, minimize effort duplication, and in that way, save time and resources.
- **After making a decision**, the platform can make all organizations aware of it and distribute it so everyone can profit from lessons learned in one unit.

## 8.5: Maximizing Organizational Learning

*“Good judgment comes from experience, and experience comes from bad judgment.”*—Fred Brooks    *“I expect you to learn to be better each day. I challenge you to look at each working day as an opportunity to learn more and, by doing so, to grow as a person.”*—L. David Marquet

The last goal is that architecture should help organizations learn quickly, stay up-to-date with emerging technologies and industry trends, and recommend technology upgrades. Learning is one of the primary daily tasks of architects. Architects must **proactively identify relevant new technology developments**. They must create pragmatic technology recommendations for concrete platforms across the organization based on their understanding of these developments.



image by istock

The Grounded Architecture structure can support learning in multiple ways:

- The **Data Foundation** can accelerate learning and adoption of new technologies by providing more data to facilitate reflection and exploration.
- The **People Foundation** can create spaces for sharing knowledge about architecture and technology. These spaces include but are not limited to regular update calls, knowledge-sharing sessions, or conferences. In addition to creating spaces as a community, it can further increase our learning value by deriving generalized insights from cross-group cases. Such spaces can **maximize personal learning**, transforming individual lessons learned into shared guidelines.
- The **Architecture Activities Platform** can accelerate learning by embedding it into processes and distributing them across the organization. By defining processes to facilitate sharing knowledge and lessons learned, we can maximize learning while creating a minimal overhead.

## 8.6: Questions to Consider

It is always essential to be thoughtful about the value and impact of your work. Ask yourself the following questions:

- *How effective is your organization’s current architectural function at scale? How valuable could principles of Grounded Architecture be to enhance its efficiency?*
- *To what extent does your organization use data to inform architectural decisions? What steps could you take to move your organization from opinion-based to more data-driven decision-making?*
- *How well-aligned are the different areas within your organization, and how does this affect your architectural function? Could the Data and People Foundations principles be utilized to improve alignment?*
- *What strategies does your organization currently have to foster organizational learning? How could the methods described in the Grounded Architecture model enhance this?*
- *How quickly can your organization adopt and utilize new technologies? How could your architecture practice accelerate this process?*
- *Consider the adaptivity of your organization’s architectural function. How could your architecture practice improve it?*
- *Reflecting on the value of the “Data Foundation” concept, how effectively is your organization tracking changes or supporting what-if scenarios analysis?*
- *What role do most senior architects play in your organization? Could their time be better utilized on strategic initiatives?*
- *How sustainable is the architectural function in your organization in the absence of a strong central team? Could implementing a Data Foundation and well-connected architects help mitigate this?*

# **Part II: Being Architect**

# 9: Being Architect: Introduction



image by borko manigoda from pixabay

**IN THIS SECTION, YOU WILL:** Get an overview of guiding principles that generalize my view on what it means to be an architect in practice.

In this part of the book, I introduce several guiding principles that generalize my view on what it means to be an architect in practice:

- **Architects as Superglue:** Architects in IT organizations should develop as “superglue,” people who hold architecture, technical details, business needs, and people together across a large organization or complex projects.
- **Skills:** A typical skillset of an architect includes hard (technical) skills, soft (people & social) skills, and business skills.
- **Impact:** Architects’ work is evaluated based on their impact on the organization. They must demonstrate that they identify, tackle, and deliver on strategic problems, have a deep and broad influence, and deliver solutions that few others can.
- **Leadership:** My view of architecture leadership is inspired by David Marquet’s work and Netflix’s valued behaviors.
- **Architects’ Career Paths: Raising the Bar:** Architects’ career paths ideally stem from a strong engineering background. Hiring architects requires constantly raising the bar to ensure a strong and diverse team structure.

# 10: Architects as Superglue



**IN THIS SECTION, YOU WILL:** Understand the view on architects as superglue (people who hold architecture, technical details, business needs, and people together across a large organization or complex projects) and get valuable tips on developing “superglue” abilities.

**KEY POINTS:**

- Architects in IT organizations should develop as “super-glue,” people who hold architecture, technical details, business needs, and people together across a large organization or complex projects.
- Architects need to be technically strong. But their unique strengths should stem from being able to relate technical issues with business and broader issues.
- Architects should stand on three legs: skills, impact, and leadership.

I believe architects in IT organizations should develop as “superglue.” I borrow the “superglue” view from [Adam Bar-Niv and Amir ShenHAV from Intel<sup>1</sup>](#). They pointed out that instead of the superhero, we need “superglue” architects - **the people who hold architecture, technical details, business needs, and people together** across large organizations or complex projects. More recently, Tanya Reilly presented a [similar view<sup>2</sup>](#) concerning software engineering positions.

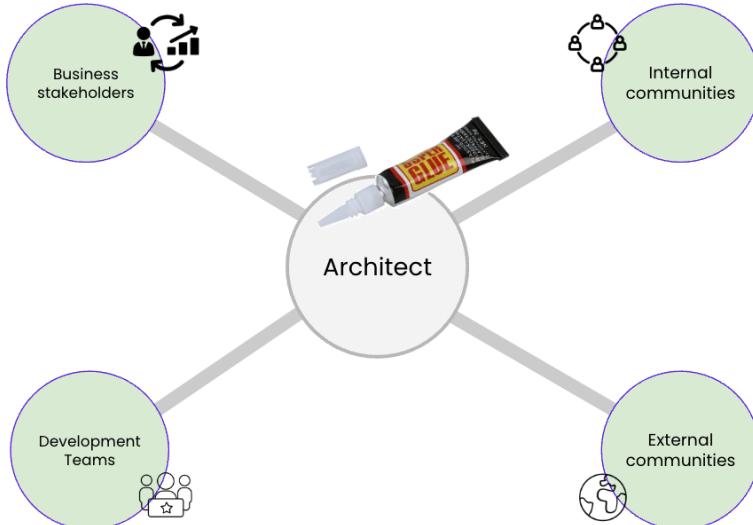
The superglue characteristics mean serving as the **organizational connective tissue**, linking the **“business wheelhouse”** and the **“engine room.”** Architects, of course, need to be technically strong. But their unique strengths should stem from being able to relate, or glue, technical issues with business and broader issues.

From discussions I’ve had with engineering leaders, engineers, and architects, Figure 1 has crystallized as a representation of the “superglue” metaphor for architects.

---

<sup>1</sup><https://saturn2016.sched.com/event/63m9/cant-find-superheroes-to-help-you-out-of-a-crisis-how-about-some-architecture-and-lots-of-superglue>

<sup>2</sup><https://noidea.dog/glue>



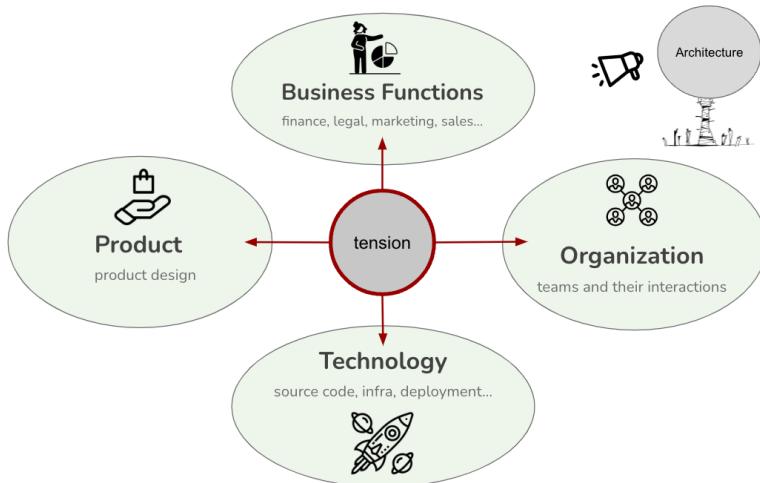
**Figure 1:** Architects serve as a superglue, connecting development teams with business stakeholders while linking teams with the internal communities and the external world.

Architects must have good relationships with developer teams, local business stakeholders, and functions. Simultaneously, such a person needs to be well-connected with broader internal communities. External visibility is essential for architects, who can bring ideas from outside into the organization and promote the organization to the outside world.

## 10.1: Supergluing in Action: Reducing Tension among Business Functions, Product, Technology, Organization

The primary value of superglue architects in complex organizations is **aligning business, product, technology, and organizational functions**. Each of these four parts has its own designs or architectures.

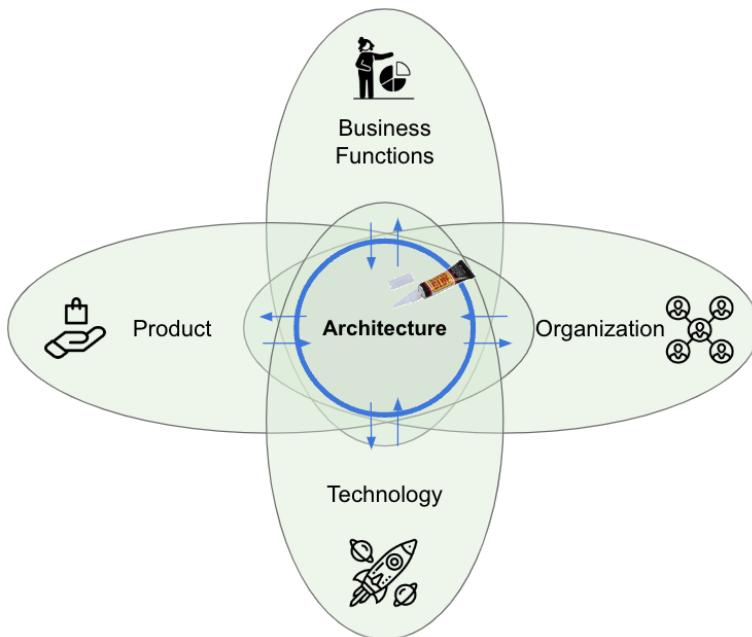
Technology, product, organization, and business functions face specific challenges. Ideally, these structures all change simultaneously and stay in perfect sync. But in practice, these structures change and move at different speeds, leading to **misalignment and tension among them** (Figure 2). For example, we may organize teams using a well-defined domain model (organizational design). Still, if our IT system is a monolith (technical design), our teams will collaborate in a different pattern than the organizational design would suggest. On the other hand, if our teams are well-aligned with the technical domain model and implementation (e.g., teams have full ownership of microservices and can deploy them independently), but the product architecture differs from the microservice modularization (e.g., product features are differently grouped than technical services supporting them), we may need to change dozens of microservices when introducing relatively simple product change. Similarly, tension occurs when business objectives are misaligned with product or technology objectives (e.g., try reducing short-term costs while adding new features and migrating to the public cloud).



**Figure 2:** The tensions between technology, product, organization, and business functions.

The main problem with this tension is that it can **slow things down** due to miscommunication or other misalignments, lead to **bad decisions** due to lack of information, introduce **unnecessary complexity**, and lead to **missed opportunities**. Too frequently, architecture sits on the side, shouting principles and abstract ideals that everyone ignores.

By acting as a **superglue**, the architecture practice can help **reduce tension** between technology, product, organization, and business functions, ensuring that critical conversations happen between these units. As Figure 3 illustrates, architecture should not try to be a superglue by adding new constructs between these four elements but by **bringing them closer together**. I sometimes joke that **architecture practice is a self-destructive function**, as by bringing these elements together, you are removing the need to have architecture practice.



**Figure 3:** Architects should be in the middle of reducing tensions between technology, product, organization, and business functions.

While staying close to technology (engine room), architects must ensure that technology is serving the needs of customers and the business and that technical architecture is well aligned with the organizational design. At the same time, architects can help ensure that business, product, and organizational designs are well-informed about the state, risks, and opportunities of an organization's technology to avoid creating impractical strategies, setting unrealistic goals, or missing opportunities. More specifically, there are several key risks that the misalignment brings, and architects need to be aware of:

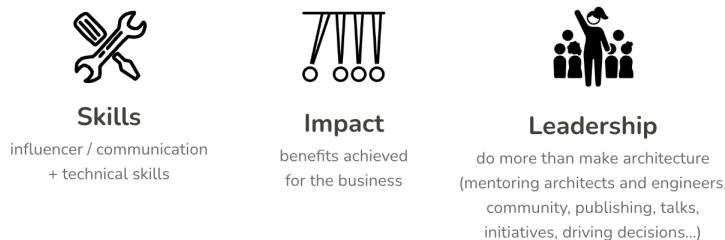
- **Building wrong products**, if technology implementation makes incorrect assumptions not aligned with product requirements.

- **Wrong prioritization of activities**, if there are no clear business and product metrics, we may build “interesting” products that do not create value for our customers and business.
- **Unexpected delivery delays** due to underestimation of complexity, effort, and dependencies,
- **Duplication of effort** due to lack of business or product harmonization needed to facilitate building shared components,
- **Building too complex products**, as we may create a complex configurable system to address all possible cases, but we could have decided to simplify and harmonize our processes and products and build a more straightforward technical solution.
- **Overengineering** due to both lack of pushback to simplify products and lack of understanding of technology (e.g., use of a complex and expensive messaging middleware capable of handling millions of messages per hour for the use case where we have a few thousand messages per day),
- **Building too simple, unscalable products**, if we made assumptions that we will simplify and harmonize our processes, but in reality, we need to keep this essential complexity and support many variations as they create value for customers and businesses.
- **Building low-quality products**, due to creating unnecessary complexity and lack of critical knowledge and expertise in the organization,
- **Having complicated dependencies between teams that slow them down** due to suboptimal organizational design and lack of awareness of all the links between systems and people,
- **Creating fragile, unsustainable team structures** (e.g., having only one or two developers in some critical technology).

## 10.2: Superglue Abilities

Setting the architects' goals to be "superglue" also requires some thought on developing architects as a superglue. Borrowing from Gregor Hohpe's view on architect development from his book Software Architecture Elevator, I share the view that our architects should stand on three legs (Figure 4):

- Skills
- Impact
- Leadership



*Figure 4: Architect Profile: Skills + Impact + Leadership.*

### 10.2.1: Skills

Architects must have proper skill sets, possessing both knowledge and the ability to apply relevant knowledge in practice. These skills should include technical (e.g., cloud architecture or Kubernetes technology) and communication and influence skills.

A typical skillset of an architect includes:

- **Hard (technical) skills**, including extensive knowledge of both new technology and legacy technology stacks,
- **Soft skills**,

- Product development knowledge,
- Business domain knowledge, and
- Decision-making skills.

The section [Skills](#) provides more details.

### 10.2.2: Impact

The impact should be measured as a benefit for the business. Architects need to ensure that what they are doing profits the business. Architects that do not make an impact do not have a place in a for-profit business.

Examples of such impact may include:

- Aligning business, product, technology, and organizational strategies,
- Process optimizations and improvements with real, measurable impact on the work of an organization,
- Cost optimizations of systems based on data-informed decisions,
- Developing pragmatic **technology strategies**, helping businesses reach goals sustainably,
- Driving **delivery of products**, supporting teams to increase quality and speed of delivery,
- Supporting **business innovation**, bringing new pragmatic ideas aligned with business strategy and goals.

The section [Impact](#) provides more details.

### 10.2.3: Leadership

Leadership acknowledges that experienced architects should do more than make architecture:

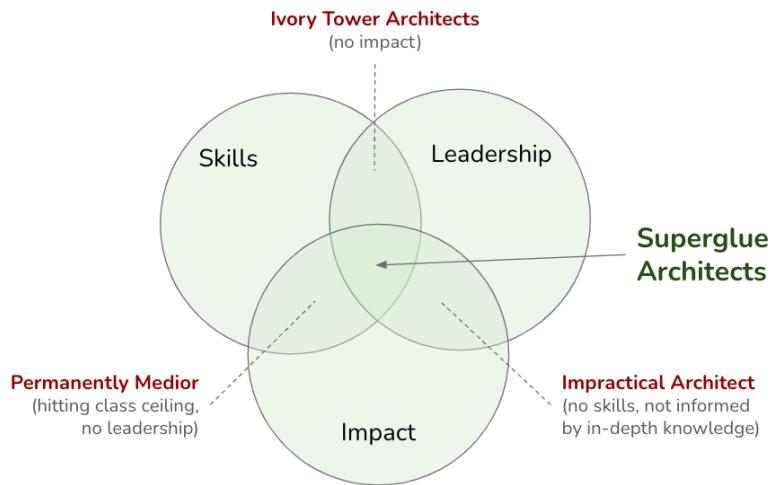
- They are a **role model for others** in the company on both the technical and cultural front.
- Their **technical influence** may extend **beyond your organization and reach the industry at large**.
- They **lead efforts** that **solve important problems** at the engineering area level.
- They may **contribute to the broader technical community** through **tech talks, education, publications, open-source projects, etc.**
- They **raise the bar of the engineering culture** across the company.

**Mentoring junior architects** is the most crucial aspect of senior architects' leadership. Feedback cycles in (software) architecture are inherently slow. Mentoring can save new architects many years of learning by doing and making mistakes.

The section [Leadership](#) provides more details.

#### 10.2.4: Balanced Development

Architects must have a **minimal “length”** of all of these “legs” to be successful (Figure 5). For instance, having skills and impact without leadership frequently leads to **hitting a glass ceiling**. Such architects plateau at an intermediate level and cannot direct the company to innovative or transformative solutions. Leadership without impact lacks foundation and may signal that you have become an **ivory tower architect** with a weak relation to reality. And having impact and leadership qualities but no skills leads to **impractical decisions** not informed by in-depth knowledge.



**Figure 5:** Architects must have a minimal “length” of all “legs” to be successful.

## 10.3: Questions to Consider

Being a superglue architect means constantly developing and re-defining your role to benefit a changing organization. Ask yourself the following questions:

- *How well do you think you currently embody the characteristics of a “superglue” architect? Which areas could you improve on to become more effective in this role?*
- *Reflect on your ability to connect the “business wheelhouse” and the “engine room” within your organization. How effectively do you bridge the gap between technical issues and business needs?*
- *How strong are your relationships with developer teams, local business stakeholders, and broader internal communities? How could you strengthen these connections?*
- *How much external visibility do you currently have? How could this be enhanced to promote the flow of ideas into and out of the organization?*
- *Can you identify specific instances of tension between your organization’s technology, product, organization, and business functions? What caused this tension, and how was it addressed?*
- *How could your current architecture aid in reducing tension between these functions?*
- *Have you witnessed the architecture sitting on the side, being ignored? If so, what steps can you take to actively involve architecture in decision-making processes?*
- *Are conversations between the technical, product, organizational, and business functions encouraged and facilitated within your organization? If not, how might they be initiated and supported?*
- *Considering the three legs of a successful architect (skills, impact, leadership), which are your strongest? Which might need more development?*

# 11: Skills

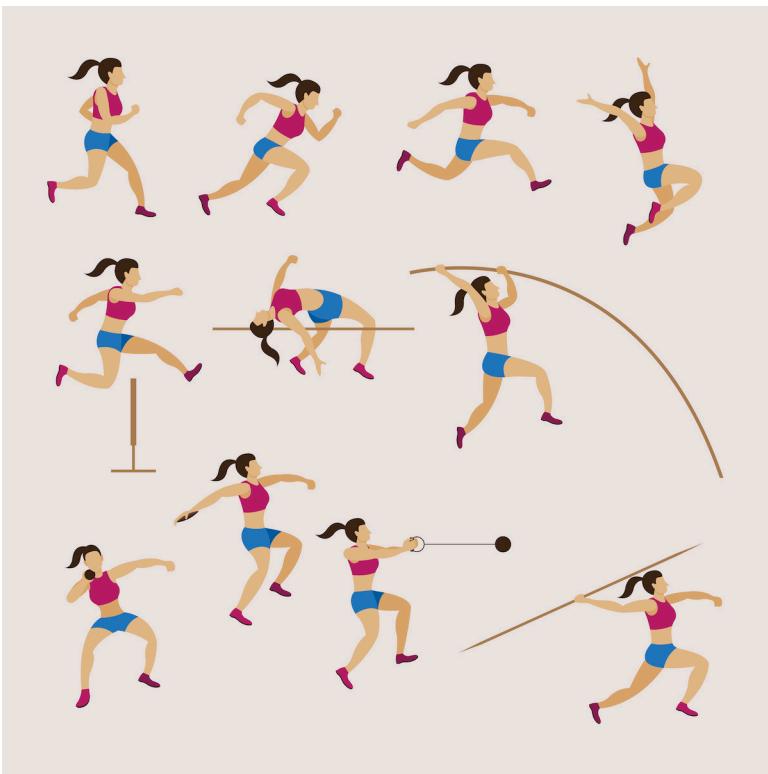


image by istock

**IN THIS SECTION, YOU WILL:** Understand that architects' skills should include a mix of technical, communication, product development, and business skills, and get valuable pointers to resources for developing these skills.

**KEY POINTS:**

- A typical skillset of an architect includes hard (technical) skills, soft (people & social) skills, product development, business skills, and decision-making skills.

Architects have to have proper skill sets. By skills, I mean possessing **relevant knowledge and the ability to apply that knowledge** in practice. These skills should include technical (e.g., cloud architecture or Kubernetes technology) and communication and influence skills.

More specifically, a typical skillset of an architect includes:

- **Hard (technical) skills**, including extensive knowledge of both new technology and legacy technology stacks,
- **Soft skills**,
- **Product development skills**,
- **Business domain knowledge**, and
- **Decision-making skills**.

## 11.1: Hard Skills

Hard or technical skills are the abilities and knowledge needed to perform specific tasks in a professional setting. In the context of technical architecture, these skills are essential for **designing**, **implementing**, and **maintaining** various aspects of an **organization's technology landscape**. Some typical hard skills that architects need in their work include (I provide links to some of the **tools**<sup>1</sup> I found helpful in obtaining these skills):

- **System design**<sup>2</sup>: System design refers to defining and developing a complex system's architecture. An architect with this skill set can create comprehensive system designs incorporating various components and sub-systems to achieve the desired functionality.
- **Engineering processes**<sup>3</sup>: An architect needs to have an in-depth understanding of engineering processes, including software development life cycle, Agile development, DevOps, and continuous delivery.
- **Design patterns**<sup>4</sup> and **tactics**<sup>5</sup>: A good architect should be familiar with design patterns and tactics such as Cloud Design Patterns, Model-View-Controller (MVC), Service-Oriented Architecture (SOA), and Microservices. These patterns help architects design modular, scalable, and maintainable systems.
- **Security and privacy by design**<sup>6</sup>: In today's world of cyber-security threats, architects need to have a deep understanding of security and privacy best practices. They must ensure that

---

<sup>1</sup><https://obren.io/tools>

<sup>2</sup><https://blog.pragmaticengineer.com/system-design-interview-an-insiders-guide-review/>

<sup>3</sup><https://obren.io/tools/catalogs/?id=design-tactics-high-performing-technology-organizations>

<sup>4</sup>[https://obren.io/tools?tag=design\\_patterns](https://obren.io/tools?tag=design_patterns)

<sup>5</sup>[https://obren.io/tools?tag=design\\_tactics](https://obren.io/tools?tag=design_tactics)

<sup>6</sup><https://obren.io/tools?tag=security>

the systems they design are secure and comply with data protection regulations.

- **System optimizations**<sup>7</sup>: An architect should know how to optimize systems for performance and scalability. They should be familiar with tools and techniques for profiling and tuning systems to achieve optimal performance.
- **Source code structures and maintainability**<sup>8</sup>: Architects should have a good understanding of software engineering principles such as clean code, code maintainability, and refactoring. They should be able to design systems that are easy to maintain and modify.
- **Reliability and stability (anti)patterns**<sup>9</sup> and **tactics**<sup>10</sup>: An architect should know the typical reliability and stability issues that can arise in complex systems. They should be able to identify and address potential problems by using patterns and tactics such as redundancy, failover, and graceful degradation.
- **Usability**<sup>11</sup>: An architect should have a good understanding of usability principles. They should design systems that are easy to use and provide a good user experience.

---

<sup>7</sup><https://obren.io/tools/catalogs/?id=design-tactics-sig-performance>

<sup>8</sup><https://obren.io/tools/catalogs/?id=design-tactics-sig-maintainability>

<sup>9</sup><https://obren.io/tools/catalogs/?id=releaseit-stability-awareness>

<sup>10</sup><https://obren.io/tools/catalogs/?id=releaseit-stability-tactics>

<sup>11</sup><https://obren.io/tools?q=usability>

## 11.2: Soft Skills

To change the architecture of a software-intensive system ensconced in a large organization, you often have to change the architecture of the organization. And ultimately, that is a political problem, not just a technical one. —Grady Booch

Soft skills, often described as non-technical or interpersonal skills, are an integral part of social architecture, as they **enable individuals to navigate and contribute to these social systems effectively**. Social architecture refers to designing and managing social systems, interactions, and relationships within an organization or community. By developing and refining soft skills, individuals can more easily adapt to changes, collaborate with others, and foster a positive work environment. Critical soft skills related to social architecture include:

- **Communication skills**<sup>12</sup>, **written**<sup>13</sup>, **visual**<sup>14</sup>, verbal (presentation), and listening skills: Effective communication involves expressing oneself clearly and understanding and empathizing with others. These skills include written, visual, verbal (presentation), and listening skills, which are crucial for building and maintaining relationships, as well as for conveying ideas and facilitating discussions.
- **Networking and collaboration skills**<sup>15</sup>: Networking involves building and maintaining diverse professional connections. Collaboration skills encompass working effectively with others, regardless of their role or seniority. These skills

---

<sup>12</sup><https://obren.io/tools?tag=consultancy>

<sup>13</sup><https://obren.io/tools/sowhat/>

<sup>14</sup><https://obren.io/tools?tag=visuals>

<sup>15</sup><https://obren.io/tools?tag=leadership>

include partnering with peers, junior and senior colleagues, managers, and executives to achieve common goals.

- **Organizational and time management skills<sup>16</sup>:** These skills involve the ability to efficiently plan, prioritize, and manage tasks, resources, and time. Effective organization and time management are crucial for meeting deadlines, achieving goals, and maintaining a healthy work-life balance. Key aspects of these skills include prioritization, goal-setting, task management, and delegation.
- **Analytical, strategic thinking, and problem-solving skills<sup>17</sup>:** Analytical skills involve assessing and interpreting complex information to make informed decisions. Strategic thinking is the capacity to envision and plan for long-term success, while problem-solving skills include identifying and addressing challenges creatively and effectively. These skills are essential for recognizing and capitalizing on unique opportunities and creating organizational value.

---

<sup>16</sup><https://obren.io/tools?tag=reflect>

<sup>17</sup><https://obren.io/tools?tag=it>

### 11.3: Product Development Skills

Product development is creating and bringing new products or services to the market. It involves the entire journey from the conception of an idea to the product's final development, marketing, and distribution. Product development encompasses various activities and stages to transform an initial concept into a tangible and market-ready offering.

Product-led companies understand that the success of their products is the primary **driver of growth and value for their company**. They prioritize, organize, and strategize around product success. Some processes specific to product development include:

- **Idea Generation:** This stage involves generating and exploring new product ideas. Ideas can come from various sources, such as market research, customer feedback, technological advancements, or internal brainstorming sessions.
- **Market Research:** Market research assesses the feasibility and potential success of the product concept. It involves gathering information about customer needs, preferences, market trends, competition, and other relevant factors to validate the product's viability in the target market.
- **Marketing and Launch:** Before launching the product, marketing strategies and plans are developed to create awareness, generate demand, and promote the product to the target market. The activities include branding, pricing, distribution, and marketing communication activities.

Architects should be familiar with product development concepts as product development requires a multidisciplinary approach involving teams from various functions such as product management, design, engineering, and marketing. The process aims to create innovative, desirable, and commercially viable products that meet customer needs and provide a competitive advantage in a market.

## 11.4: Business Skills

Regardless of their technical or design expertise, architects must have a **solid understanding of business processes** to effectively contribute to an organization's success. Essential business skills for architects include:

- **General business concepts knowledge:** A fundamental understanding of general business concepts is essential for architects to make informed decisions and effectively communicate with stakeholders. Familiarity with finance, marketing, sales, operations, and strategy can provide a strong foundation for architects to engage with various aspects of an organization. *The Personal MBA*<sup>18</sup> book has been my favorite resource for getting familiar with such concepts.
- **Specific business domains**<sup>19</sup> of the organization: Besides general business concepts, architects should also develop a deep understanding of the specific business domain in which their organization operates. This knowledge may include knowledge of industry-specific regulations, market trends, customer preferences, competitive landscape, and more. Gaining insights into the specific business domain enables architects to better align their work with the organization's goals, strategies, and priorities.
- **Business analysis and requirements gathering:** Architects should be adept at analyzing business needs and gathering requirements from various stakeholders. This skill involves understanding the organization's objectives and translating them into functional and technical specifications that can guide the design and development of solutions.

---

<sup>18</sup><https://personalmba.com/>

<sup>19</sup>[https://obren.io/tools?tag=domain\\_models](https://obren.io/tools?tag=domain_models)

## 11.5: Decision-Making Skills

Architects' work always requires pragmatic decision-making. Decisions are the steering wheel of organizations. Architects outside of key decisions will have a limited impact on the organization.

Architects will have two roles concerning decision-making:

- Be actual decision-makers,
- Be advisors to decision-makers. and
- Evaluate and provide feedback on the decisions of others.

Essential decision-making skills include:

- Understanding that a decision is an irrevocable allocation of resources: money, human effort, time, physical actions, and opportunities.
- Know the basic of **decision intelligence** - the discipline of turning information into better action.
- Understand key problems of poor decision-making, such as the **outcome bias**, and how to deal with it.
- Know when to use **intuition**, and the value of clairvoyance.
- Understand that there is no such thing as not making a decision. You are more likely making the implicit decision to delay, postpone, or deprioritize a decision.

## 11.6: Questions to Consider

- *On a scale from 1 to 10, how would you rate your current architectural skill sets, considering technical, communication, product, business skills, and decision-making skills?*
- *Reflect on your technical skills. How proficient are you in system design, understanding engineering processes, recognizing design patterns and tactics, ensuring security and privacy, optimizing systems, and maintaining code structures?*
- *Do you need to develop specific hard skills to enhance your architectural performance?*
- *How effectively do you communicate (in writing, visually, verbally, and through listening)? How strong are your networking and collaboration skills, and how well do you manage your time and organizational tasks?*
- *Can you identify an instance where your problem-solving skills and strategic thinking have significantly influenced your work as an architect?*
- *Looking at business skills, how well do you understand general business concepts, and how familiar are you with the specific business domain of your organization?*
- *How competent are you in business analysis and requirements gathering? Can you share an example where you effectively translated business objectives into functional and technical specifications?*
- *Are there any soft or business skills you need to develop or improve to succeed in your role as an architect?*
- *Reflect on how you have used your soft skills to effect organizational change. Are there areas or situations where you could have applied these skills more effectively?*
- *How do you balance developing and maintaining your hard, soft, and business skills? Is there a particular area you tend to focus on more, and why?*

## 12: Impact



image by wikipedia / sandi morris.

**IN THIS SECTION, YOU WILL:** Understand that architects' work is evaluated based on their impact on the organization and get guidelines for making an impact.

**KEY POINTS:**

- Architects' work is evaluated based on their impact on the organization.
- Architects can make an impact via three pillars: Big-Picture Thinking, Execution, and Leveling-Up.

Architects' work is evaluated based on **their impact on the organization**. Architects typically make an impact by:

- **Identifying, tackling, and delivering on strategic problems** at the organization and area levels (domain or technical areas). Architects' work needs to be prioritized based on global strategic objectives.
- **Having a deep and broad influence** on a domain, product, or technology area. Architects sometimes need to go deep, addressing specific critical issues in one area. And frequently, that needs to look broad, creating impact by leveraging the results across multiple teams.
- **Delivering solutions that few others can**, sometimes by their heavy lifting but more often by their ability to orchestrate extensive group efforts. Architects can help move the organization forward by leveraging their hard technical skills and soft strategic, execution, and people skills.

To make this impact, architects need a few key competencies.

## 12.1: Pillars of Impact

Architects must have **strong technical, people, and business skills**, ideally obtained through years of practice. On top of this **strong foundation**, architects need to develop competencies that enable them to use their experiences and abilities to **impact organizational performance positively**. The more senior architects become, the more their competency development should be driven by the impact they need to have rather than mere skills development. I typically coach architects in the context of concrete activities, guiding their development via involvement in the proper set of actions and crafting skills developments based on challenges in making an expected impact in practice.

I use Staff Engineering roles as an inspiration for the development of architects. Tanya Reilly's book [The Staff Engineer's Path](#)<sup>1</sup> and Will Larson's book [Staff Engineer: Leadership beyond the management track](#)<sup>2</sup> are helpful guides in defining the responsibilities of architects.

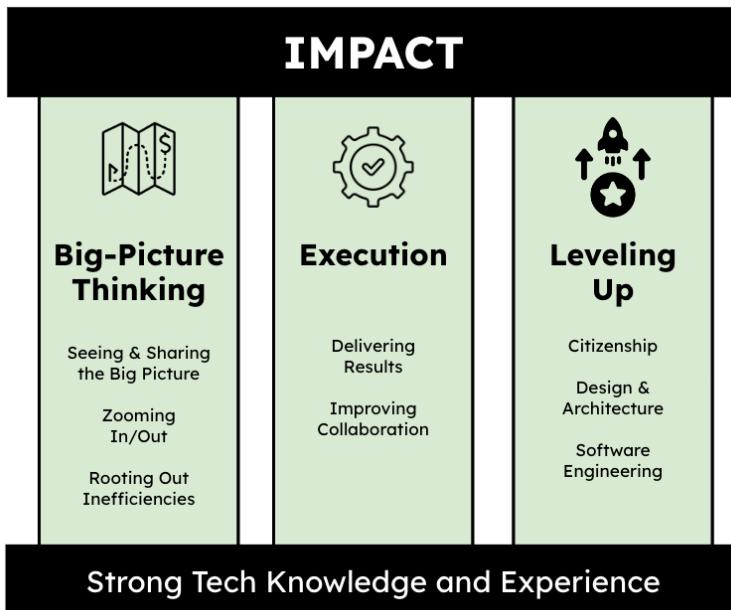
Inspired by The Staff Engineer's Path by Tanya Reilly, I group architects' impact-making competencies into three groups (Figure 1):

- Big-picture thinking,
- Execution, and
- Leveling up.

---

<sup>1</sup><https://www.oreilly.com/library/view/the-staff-engineers/9781098118723/>

<sup>2</sup><https://staffeng.com/guides/staff-archetypes/>



*Figure 1: Key competencies of architects. Inspired by The Staff Engineer's Path by Tanya Reilly.*

### 12.1.1: Big-Picture Thinking

Architects are frequently the only people in the organization with a “helicopter view,” overseeing vast domains and being able to foresee the consequences of decisions in a broader context. As big-picture thinkers, architects can help organizations in multiple ways:

- Seeing the **big picture** can identify high-leverage points for maximum impact.
- **Helping others to see the big picture** and create tools (e.g., [Data Foundation](#)) that facilitate big-picture thinking.
- Being able to **zoom in and out**, having a strategic overview, but being able to go deep and engage with implementation details.

- Using big-picture thinking to **consistently root out inefficiencies** and lead the adoption of technologies and processes that **make multiple teams more efficient**.

### 12.1.2: Execution

As execution-focused practitioners, architects must be able to help deliver results and improve collaboration.

Architects can help deliver results by combining their skills with a high dose of pragmatism:

- **Create meaningful solutions** rather than theoretical ideals and models.
- **Break down complex problems** to enable the delivery of impactful results.
- **Craft pragmatic plans** by considering technical, logistical, and organizational constraints.

Architects also can help execution by finding ways to enable others to collaborate and work better:

- **Creating alignment** and **improving collaboration** within their areas or the wider engineering organization.
- **Collaborate meaningfully across groups** to build trust and increase execution speed.

### 12.1.3: Leveling Up

Many frequently see architects as leaders and role models that should help organizations to raise the bar on the technical and cultural fronts. I group such role of an architect into three categories: citizenship, design and architecture, and software engineering.

### **12.1.3.1: Citizenship**

Architects should look broader and raise the bar of practices and behaviors in their organizations:

- **Contribute to the broader technical community** through tech talks, education, publications, open-source projects, etc.
- **Have influence that extends** beyond their organization and reaches the industry at large.
- **Lead efforts** that solve **important problems** in their areas.
- **Raise the bar** of the engineering culture across the company.

### **12.1.3.2: Design and Architecture**

Architects are leading authority for systematic and strategic design and architecture topics:

- **Identify and solve systemic architectural problems.** Architects quickly recognize systemic problems and can **articulate possible solutions** to them.
- **Improve the definition of best practices** and architecture with deep domain knowledge.

### **12.1.3.3: Software Engineering**

Lastly, architects can help by staying well-connected to software engineering practice, leveraging their experience to:

- **Promote and demonstrate best-in-class** of code, documentation, testing, and monitoring practices.
- **Solve challenging technical and execution problems** that few others can.

## 12.2: Questions to Consider

- Reflect on the impact you have had on your organization. Have you prioritized your work based on global strategic objectives, and what has been the outcome?
- Can you identify instances where you had to go deep into a specific issue and others where you needed a broad perspective across multiple teams? How did you manage both scenarios?
- How have you used your technical, strategic, execution, and people skills to deliver solutions? Can you share an example?
- How can you build on your technical, people, and business skills to positively impact your organization's performance? How do you measure this impact?
- As an architect, how can you develop your big-picture thinking ability? Can you give an example of how your big-picture thinking helped to identify a high leverage point for maximum impact?
- Reflect on your role in execution. How can you help in delivering results and improving collaboration? Can you share an example where your pragmatism resulted in a meaningful solution?
- What initiatives could you have taken to improve collaboration and build trust within your organization?
- Have you contributed to the broader technical community through tech talks, education, publications, open-source projects, etc.?
- How could you help solve significant problems in your area and raise the bar of the engineering culture across the company?
- Can you provide an example of a systemic architectural problem you identified and the solution you proposed?
- How would you promote and demonstrate best-in-class practices in coding, documentation, testing, and monitoring?

# 13: Leadership



image by david mark from pixabay

**IN THIS SECTION, YOU WILL:** Understand how to apply ideas from David Marquet's work and Netflix's valued behaviors to develop architects' leadership traits.

**KEY POINTS:**

- My view of architecture leadership is inspired by David Marquet's work and Netflix's valued behaviors.
- Marquet focused on leadership and organizational management, particularly emphasizing the principles of Intent-Based Leadership.
- Borrowing from Netflix's original values, the following behavioral traits are crucial for architects: communication, judgment, impact, inclusion, selflessness, courage, integrity, curiosity, innovation, and passion.

My approach to architecture leadership draws inspiration from two sources: **David Marquet's<sup>1</sup> leadership principles**, as articulated in his book “Turn the Ship Around!” and Netflix's valued behaviors. Marquet's ideas emphasize empowering team members, providing clarity, decentralizing decision-making, striving for continuous improvement, and practicing servant leadership. Meanwhile, **Netflix's valued behaviors<sup>2</sup>** offer useful guidance for coaching and developing architects aligned with the “super glue” version.

---

<sup>1</sup><https://davidmarquet.com/>

<sup>2</sup><https://jobs.netflix.com/culture>

## 13.1: David Marquet's Work: The Leader-Leader Model

Marquet's work is closely tied to **the Leader-Leader model of leadership**, a leadership style where authority is shared across a team or organization instead of being concentrated at the top. In this model, every team member has something valuable to contribute and can work together toward the group's success.

This leadership approach empowers individuals to **take ownership of their work and collaborate** with others to achieve common goals. Instead of relying on a single leader to make all decisions, authority and responsibility are distributed across the team.

The leader-leader model is an excellent standard for architects' leadership vision. Like managers in a leader-leader model, **architects should act more as facilitators, coaches, and mentors** than traditional top-down decision-makers. They provide team members guidance, support, or resources to help them achieve their goals and reach their full potential.

One of the key benefits of a leader-leader model is that it creates a **more collaborative and inclusive work environment**. It allows individuals to contribute their unique perspectives, experiences, and skills to the group, promoting ownership and accountability for the team's success. This model also **helps build trust and more robust team relationships**, increasing productivity, creativity, and innovation.



image by istock

David Marquet's book "Leadership is Language" provides practical advice for leaders looking to create a more collaborative, innovative, and inclusive organizational culture. He emphasizes the importance of language and communication in leadership and introduces the phrase "**I intend to**" as a powerful tool for clarifying intent and **empowering team members** (Figure 1). When team members give intent, the psychological ownership of those actions shifts to them, making them the originators of thought and direction instead of passive followers. This shift in language helps to promote a more collaborative work environment.

I have found a phase "**I intend to**" to be a powerful catalyst for positioning architecture work. The phrase helps describe the **work architect as someone others expect to take the initiative and lead efforts**. But also to describe the desired interaction of architects with the teams they work with, where we hope teams share their intentions which architects can help improve.

LEADER	LEADER
7. I've been doing...	7. What have you been doing?
6. I've done...	6. What have you done?
5. I intend to...	5. What do you intend?
4. I would like to...	4. What would you like to do?
3. I think...	3. What do you think?
2. I see...	2. What do you see?
1. Tell me what to do.	1. I'll tell you what to do.
WORKER	BOSS

**Figure 1:** Leadership language. Based on Intent-Based Leadership, by David Marquet.

## 13.2: Netflix's Valued Behaviors: Leadership Behaviors

The [Netflix overview of their valued behaviors](#)<sup>3</sup> is a leading inspiration for how I coach and develop architects. The following sections summarize these behaviors, borrowing from the Netflix original values but rearranging them in the order I see as more relevant for architects.

### 13.2.1: Communication

Architects can only be successful if they are effective communicators. More specifically, as an architect, you need to have the following communication traits:

- You are **concise** and articulate in **speech and writing**
- You **listen well** and **seek to understand** before reacting
- You maintain **calm poise** in **stressful situations** to draw out the clearest thinking
- You **adapt your communication style** to work well with people from around the world who may not share your native language

### 13.2.2: Judgment

People frequently call architects to be objective judges when others cannot agree or need an objective second opinion. As an architect, you'll be able to make sound judgments if:

- You are good at using **data to inform your intuition**

---

<sup>3</sup><https://jobs.netflix.com/culture>

- You make wise **decisions** despite **ambiguity**
- You identify **root causes** and go beyond treating symptoms
- You make decisions based on **the long-term**, not near-term
- You **think strategically** and can articulate what you are, and are not, trying to do

### 13.2.3: Impact

As discussed in the [Architects as Superglue](#), the architect's impact should be measured as a benefit for the business. Architects need to ensure that what they are making profits the company. As an architect, you need to show the following impact traits:

- You accomplish significant amounts of **important work**
- You **make your colleagues better**
- You focus on **results over processes**
- You demonstrate **consistently** strong performance so **colleagues can rely upon you**

### 13.2.4: Inclusion

Architects must work with many different people and groups inclusively. You will be able to do so if:

- You **collaborate effectively** with people of **diverse backgrounds and cultures**
- You **nurture and embrace differing perspectives** to make better decisions
- You **focus on talent and values** rather than a person's similarity to yourself
- You are **curious about how our different backgrounds affect us** at work rather than pretending they don't affect us

### 13.2.5: Selflessness

Architects always need to consider the best interests of their organizations. This broader view is essential in group conflicts to enable resolutions that benefit the organization. To be able to operate in such a way, you need to show the following selflessness traits:

- You seek what is **best for your organization** rather than what is best for yourself or your group
- You **share information openly and proactively**
- You **make time to help colleagues**
- You are **open-minded** in search of the best ideas

### 13.2.6: Courage

Being an architect is not always a comfortable position as you will need to be a part of difficult decisions many will not be happy about. You need to have enough courage to make such difficult calls. You will be able to do so if you show the following traits:

- You **say what you think** when it's in the best interest of your organization, even if it is uncomfortable
- You are willing to be **critical of the status quo**
- You make **tough decisions** without agonizing
- You **take smart risks** and are open to possible failure
- You **question actions inconsistent** with the organization's values
- You **can be vulnerable** in search of truth

### 13.2.7: Integrity

Architects need to operate as trusted advisors. Integrity is essential for such a position of architects. To perform successfully as trusted advisor, you need to show the following traits:

- You are known for **honesty**, authenticity, **transparency**, and being **non-political**
- You only say things about fellow employees that you **speak to their face**
- You **admit mistakes** freely and openly
- You **treat people with respect** independent of their status or disagreement with you

### 13.2.8: Curiosity

As architects, we must proactively identify relevant new technology developments. Based on our understanding of these developments, we must create pragmatic technology recommendations for concrete platforms across the organization. That means that as an architect, you need to stay curious:

- You **learn rapidly and eagerly**
- You **make connections** that others miss
- You seek **alternate perspectives**
- You **contribute effectively** outside of your specialty

### 13.2.9: Innovation

More than curiosity is required. To make an impact as an architect, you need to create helpful innovations:

- You create **new ideas** that prove useful
- You keep your organization nimble by **minimizing complexity** and finding time to simplify
- You **re-conceptualize issues** to discover solutions to **hard problems**
- You **challenge prevailing assumptions** and suggest better approaches
- You **thrive on change**

### 13.2.10: Passion

Architects are frequently role models for others. As such, you need to show the following traits:

- You **inspire others** with your thirst for excellence
- You **care intensely** about your customers and organization's success
- You are **tenacious and optimistic**
- You are **quietly confident and openly humble**

### 13.3: Questions to Consider

- Reflect on the Leader-Leader model of leadership model in your work. How can you empower your team members and encourage them to take ownership of their work?
- Have you acted as a facilitator, coach, or mentor as an architect? Can you share an example of when you gave team members guidance, support, or resources to achieve their goals?
- How does the phrase “I intend to” resonate with your approach to architecture work? How can it change your perspective on taking the initiative and leading efforts?
- How effective do you believe your communication skills are?
- How can you foster an inclusive working environment as an architect? How do you nurture and embrace differing perspectives to make better decisions?
- Reflect on a situation where you made a decision that was best for the organization rather than what was best for yourself or your group. What was the outcome?
- Have you ever had to take an uncomfortable stance but in your organization’s best interest?
- How do you maintain integrity as a trusted advisor in your organization? Can you share an example where your honesty, authenticity, and transparency were vital?
- How have you maintained your curiosity in your role as an architect? Can you share an instance where your learning eagerness led to a significant outcome?
- What innovative solutions have you created as an architect? How have these innovations benefitted your organization?
- How do you inspire others with your passion for excellence? Can you share an instance where your optimism and tenacity led to a successful outcome?

# 14: Architects' Career Paths: Raising the Bar

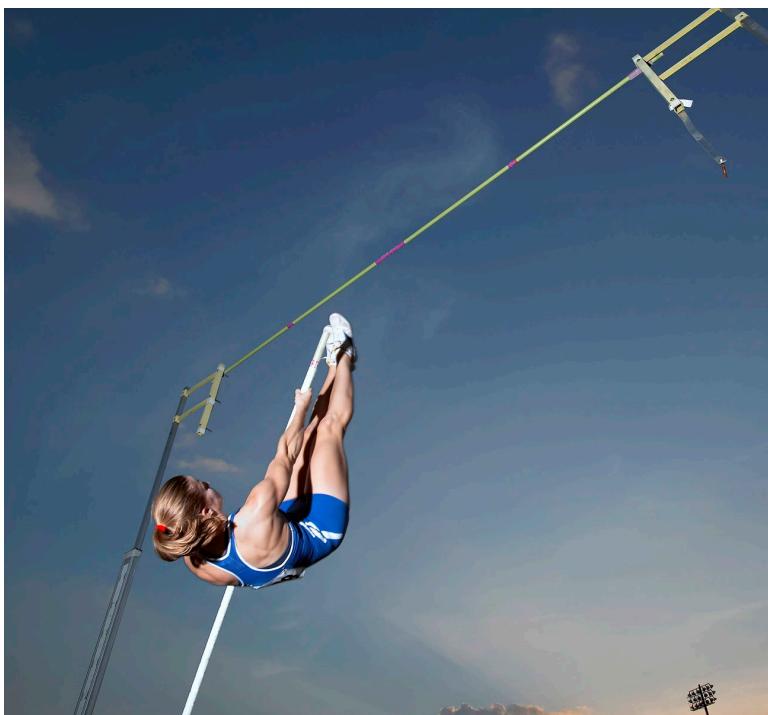


image by istock tableathny (cc by 2.0)

**IN THIS SECTION, YOU WILL:** Get ideas and tips about developing architects' career paths.

**KEY POINTS:**

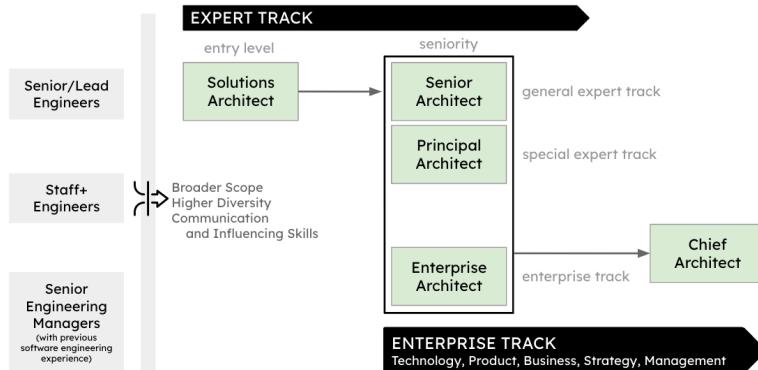
- Architects' career paths ideally stem from a strong engineering background.
- Hiring architects requires constantly raising the bar to ensure a strong and diverse team structure.

Hiring and developing architects will differ significantly per organization. Nevertheless, here I share some of the ideas and lessons learned.

My view of architecture has a **strong engineering bias**. Architects' career paths ideally stem from a **strong engineering background**. While there may be exceptions, an architect without significant real-world exposure to software engineering challenges cannot obtain enough practical knowledge to make technology decisions and build relations with developer teams.

## 14.1: Typical Architect's Career Paths

Regarding career progression, Figure 1 shows an example of architectural career paths in relation to engineering, which I used to define architectural career paths.



*Figure 1: An example of an IT architect career paths in relation to engineering.*

Stepping from an engineering position to an architecture requires three changes:

- Getting a **broader scope** of work,
- Having a **higher diversity** of work, and
- **Changing skills**, as communication and influencer skills become crucial to success.

All architects are responsible for the direction, quality, and approach within some critical area. They must combine in-depth knowledge of technical constraints, user needs, and organization-level leadership.

After the role of an Architect, I usually envision three tracks of progression:

- **Senior Architects**, generalists with broader responsibilities who can dig deep into complex issues and identifies a suitable course of action. They often navigate from one critical area to another, guided by the organization's direction.
- **Principal Architects**, senior architects with a particular focus on some area of strategic interest for an organization (e.g., data, distributed systems, frontend).
- **Enterprise Architects**, being closer to product, management, strategy, and business functions, frequently serving as senior engineering leaders' right hand.

But an architect's path can take many different directions and have many other names. More important than a formal title is a continuous search for staying relevant and making an **impact**.

## 14.2: Hiring Architects

Developing and hiring architects requires **constantly raising the bar** to ensure a strong and diverse team structure. Having more architects does not necessarily lead to a better team. Having good **alignment and diversity of perspectives** is even more important for an architecture team than for other groups.

It is vital to take more **active ownership of hiring architects**. Due to the vast diversity of how different companies define the architect's role, recruiters may need help understanding the role's requirements.

While you will need to design your hiring process, the hiring process should ensure a solid evaluation of the candidates:

- **Technical skills:** An architect must possess a solid technical background in the relevant areas, such as software development, infrastructure, cloud computing, and security. The process can assess their expertise through technical questions, tests, or case studies.
- **Communication and collaboration skills:** Architects often work with stakeholders, including business leaders, developers, and project managers. Therefore, the process could evaluate the candidate's ability to communicate effectively, work in a team, and manage stakeholders.
- **Leadership and problem-solving abilities:** As a senior team member, an architect should have strong leadership skills and the ability to solve complex problems. The process could assess the candidate's experience leading teams, making critical decisions, and resolving technical challenges.
- **Cultural fit:** The process could also evaluate the candidate's fit with the company's culture, values, and mission. The cultural fit is vital to ensure the candidate shares the same vision and will likely thrive in the organization.

In terms of steps, I typically work with some version of the following process (after standard recruitment screening):

### **Step 1: Initial Screening Interview with Chief Architect**

- Typical duration 60 min
- In this step, assessing the candidate's overall fit for the role is crucial, determining whether they possess the necessary skills, experience, and qualifications.
- Overall, the initial screening aims to identify the most promising candidates who possess the necessary skills, experience, and fit for the role of a senior solutions architect and who should proceed to the next stage of the interview process.
- Extra focus on:
  - Cultural fit
  - Leadership and problem-solving abilities

### **Step 2: In-Depth Interview with Senior/Principle/Enterprise Architects**

- Typical duration 90 min
- Extra focus on:
  - Evaluating the candidate's technical skills
  - Assessing the candidate's communication and collaboration skills
  - Understanding the candidate's leadership and problem-solving abilities

### **Step 3: In-Depth Interview with Architects and Senior Engineers**

- Typical duration 90 min

- Preparation:
  - A document describing a recent solution architecture of a candidate, providing the content for discussion and helping estimate the candidate's written skills.
  - (Optional) open-source code review of a candidate
- Extra focus on:
  - Any topics identified during Step 2 as areas that needed to explore further.

For senior positions, I typically introduce an additional step of meeting senior leadership:

#### **Step 4: Non-technical stakeholders evaluation**

- Interview with Engineering Leaders
- Interview with Product and Business Function Leaders (e.g., CPO, CMO, CFO)
- Interview with a CTO
- Extra focus on:
  - Leadership abilities
  - Communication and collaboration skills

With the described steps, you can get a solid overview of all critical aspects of superglue architects. In particular, the involvement of people outside architecture or engineering is crucial to minimize risk related to a lack of interest and ability to engage with all relevant stakeholders.

### 14.3: Questions to Consider

- *Reflect on career paths in architecture. How can an engineering background impact effectiveness of an architect?*
- *Reflect on your career progression in architecture. How can you continuously stay relevant and make an impact in your role?*
- *If you were involved in the hiring process for architects, how would you assess a candidate's technical skills, communication and collaboration skills, leadership and problem-solving abilities, and cultural fit?*
- *What strategies would you implement to ensure you continuously raise the bar in developing and hiring architects in your organization?*
- *How could you demonstrate your communication and collaboration skills as an architect? Can you share an instance where these skills are crucial?*
- *How would you describe your leadership and problem-solving abilities? Can you share an example of how you've used these skills in your work?*
- *Reflect on the cultural fit between you and your organization. How do your values align with those of the company?*
- *What steps would you include in your hiring process for architects to ensure a solid evaluation of the candidates?*
- *How would you ensure diversity of perspectives within your architecture team, and is this important?*

# **Part III: Doing Architecture: Inspirations**

# 15: Doing Architecture: Introduction



image by enrique meseguer from pixabay

**IN THIS SECTION, YOU WILL:** Get a summary of the articles about doing architecture.

In the following sections, I will introduce several resources that I use as inspiration for running the Grounded Architecture practice in complex organizations. I focus on several topics not typically covered in IT architecture literature, drawing inspiration from different sources, including social sciences, economics, behavioral sciences, product management, and political sciences:

- **The Culture Map: Architects' Culture Mindfield Compass:** In multinational organizations, architects will work with many different cultures. The work of Erin Meyer, *The Culture Map*, is a beneficial tool for architects to work harmoniously with people from various cultures and backgrounds.
- **Managing Organizational Complexity: Six Simple Rules:** Six Simple Rules emphasize that, in today's complicated business environments, you need to set up organizational structures based on cooperation. To deal with complexity, organizations should depend on the judgment of their people and on these people cooperating to utilize the organization's capabilities to cope with complex problems.
- **Understanding Product Development:** When it comes to product development, I generally recommend two resources for architects: "Escaping the Build Trap: How Effective Product Management Creates Real Value" by Melissa Perri and "The Discipline of Market Leader," by Michael Treacy and Fred Wiersema.
- **Architecture Governance: Mandates, Taxation, Nudge:** I promote a technology governance model combining three governing styles: nudging; taxes (economic incentives); mandates, and bans.
- **Economic Modeling: ROI and Financial Options:** I sketch two answers to the question of the economic value of technology investments and architecture: the return on investment metaphor and the financial options metaphor.
- **Decision Intelligence in IT Architecture:** Decision intelligence is the discipline of turning information into better

actions. The future of IT architecture will be closely related to decision intelligence.

- **The Human Side of Decision-Making** Decision-making is a human activity subject to human biases and limitations. Fundamental biases influencing decision-making include outcome, hindsight, and confirmation biases.

# 16: The Culture Map: Architects' Culture Compass



image by maik from pixabay

**IN THIS SECTION, YOU WILL:** Get an introduction to The Culture Map, a helpful tool for architects to work harmoniously with people from various cultures and backgrounds.

**KEY POINTS:**

- I have found the work of Erin Meyer, The Culture Map, to be a beneficial tool for architects to work harmoniously with people from various cultures and backgrounds.
- Meyer's model contains eight scales, each representing a key area, showing how cultures vary from extreme to extreme: Communicating, Evaluating, Persuading, Leading, Deciding, Trusting, Disagreeing, and Scheduling.

In multinational organizations, architects will need to work with many different cultures. The work of Erin Meyer, The Culture Map, is a beneficial tool for working harmoniously with people from various cultures and backgrounds. **Awareness of cultural differences** is even more important for architects, as they are bridging diverse cultures and domains (technology, business, product, organization).

Meyer's model contains **eight scales**, each representing a key area, showing how cultures vary from extreme to extreme. The eight scales describe a continuum between the two ends which are diametric opposite or competing positions:

- **Communicating** – Are cultures low-context (simple, verbose, and clear) or high-context (rich deep meaning in interactions)?
- **Evaluating** – When giving negative feedback, does one give it directly or prefer being indirect and discreet?
- **Persuading** – Do people like to hear specific cases and examples or prefer detailed holistic explanations?
- **Leading** – Are people in groups egalitarian or prefer hierarchy?

- **Deciding** – Are decisions made in consensus or made top-down?
- **Trusting** – Do people base trust on how well they know each other or how well they work together?
- **Disagreeing** – Are disagreements tackled directly, or do people prefer to avoid confrontations?
- **Scheduling** – Do people see time as absolute linear points or consider it a flexible range?

It is essential to highlight that a culture map is a framework used to understand and **compare cultural differences in a nuanced and respectful way**. It aims to highlight the diverse ways people communicate, work, and interact. Unlike stereotypes, which are often oversimplified and fixed ideas about a group of people, culture maps **recognize the complexity and variability within cultures**.

Consequently, while cultural generalizations, like the culture map, can be helpful, it is crucial to realize that they are **just that - generalizations**. Many individuals from a particular culture will not fit neatly into these categories, and there can be **significant variation** within a single culture. It is best to approach cultural differences with an open mind and a willingness to learn.

I experience the culture map as enriching and broadening my interactions with people. Where I would previously be shocked by others' behavior, the culture map reminds me that I may be interpreting other actions too constrained by my cultural background.

## 16.1: Communicating

Architects need to be **good communicators**<sup>1</sup>. But what do we mean when saying someone is a **good communicator**? The responses differ wildly from society to society.



image by istock

Meyer compares cultures along the **Communicating scale** by measuring the degree to which they are **high- or low-context**, a metric developed by the American anthropologist Edward Hall (Figure 1).

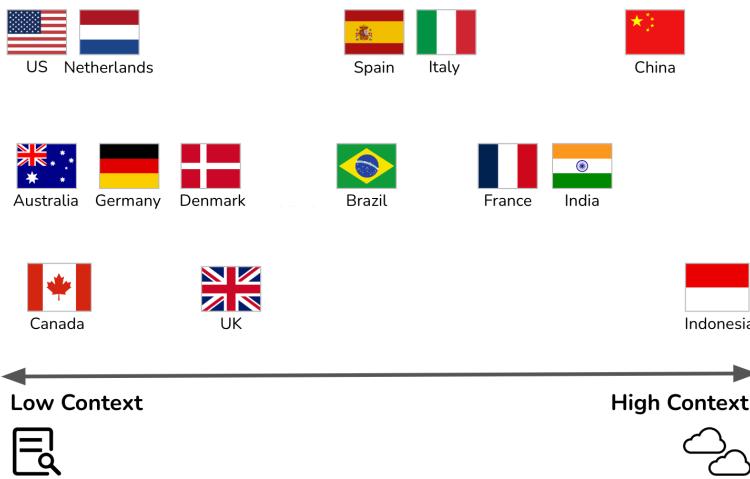
In **low-context** cultures, good communication is **precise, simple, explicit, and clear**. People take messages at face value. Repetition, clarification, and putting messages in writing are appreciated.

In **high-context** cultures, communication is **sophisticated, nuanced, and layered**. Statements are often not plainly stated but implied. People put less in writing, more is left open to interpretation, and understanding may depend on reading between the lines.

---

<sup>1</sup><https://architectelevator.com/strategy/complex-topics-stick/>

Architects should be able to **understand and adapt to different communication styles**. But when actively communicating, I find it crucial that architects provide low-context explanations. Architects will deal not only with the diverse cultural backgrounds of people but with different professional communities (technology, product, marketing, sales, finance, strategy), each with their own specific cultures and buzzwords. To bridge such diverse communities, communicating in a culture-sensitive and buzzword-free way is a valuable skill for any architect.



**Figure 1:** The Communicating scale: a graphical representation of some Culture Map countries (the cultural context of my recent professional interactions).

## 16.2: Evaluating

Architects need to **provide constructive criticism** of the plans and ideas of others. All cultures believe that people should give criticism constructively, but the definition of “constructive” varies greatly.

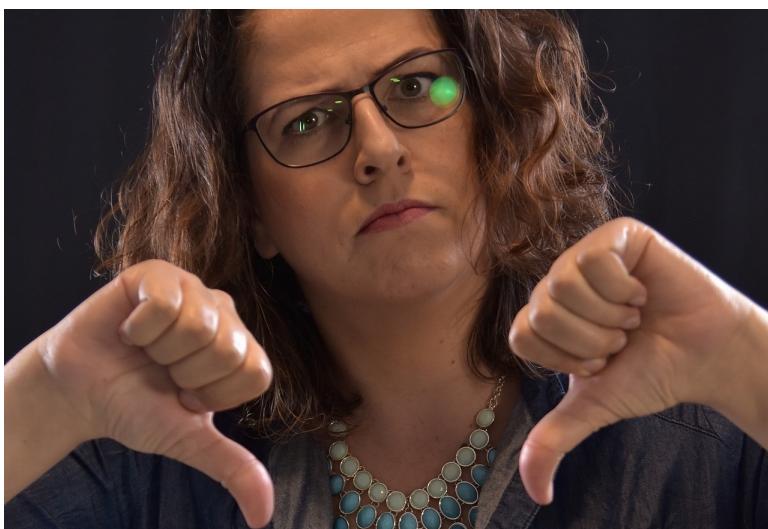
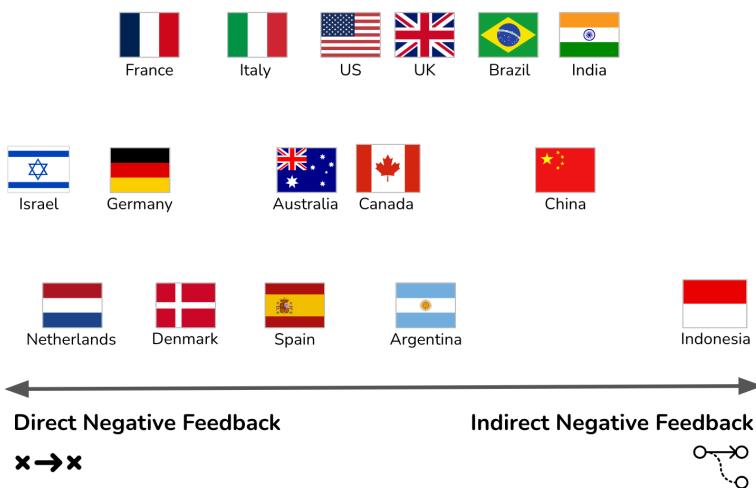


image by rickey123 from pixabay

The Evaluating scale measures a preference for **frank versus diplomatic feedback**. Evaluating is different from the Communicating scale; many countries have different positions on the two scales. According to Meyers, the French are high-context (implicit) communicators relative to Americans. Yet they are more direct in their criticism. Spaniards and Mexicans are at the same context level, but the Spanish are much franker when providing negative feedback (Figure 2).

Providing constructive **criticism in the right way** is crucial for architects to make any impact. Sometimes the same feedback will lead to different reactions, even within the same teams with

members from diverse backgrounds. Being too positive in some cultures leads to underestimation of the significance of the feedback. Being too negative may result in pushback and rejection. In my experience, architects need to adapt their feedback to the audience and do lots of “**duplication**” by presenting the same feedback differently to diverse groups.



**Figure 2:** *The Evaluating scale: a graphical representation of some Culture Map countries (the cultural context of my recent professional interactions).*

## 16.3: Persuading

Architects frequently need to persuade others about decisions and plans. How you **influence others and the arguments people find convincing** are deeply rooted in culture's philosophical, religious, and educational assumptions and attitudes.

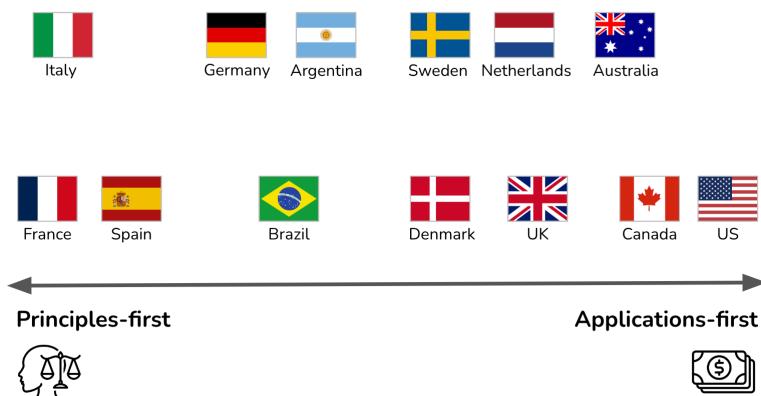


image by istock

One way to compare countries along the Persuading scale is to assess how they balance **holistic and specific thought patterns**. According to Meyers, a Western executive will break down an argument into a sequence of distinct components (specific thinking). At the same time, Asian managers tend to show how the pieces fit together (holistic thinking). Beyond that, people from southern European and Germanic cultures tend to find **deductive arguments** (principles-first arguments, building the conclusion from basic premises) most persuasive. In contrast, American and British managers are more likely to be influenced by **inductive, applications-first logic** (Figure 3).

Architects must be able to persuade in both applications-first

**and principles-first ways.** In addition to cultural differences, the additional complication comes from talking to diverse audiences. For instance, C-level executives typically have less time and may prefer applications-first presentations (“get to the point, stick to the point”). While in other parts of the company, you may need to spend a long time carefully building the argument following the principle first approach. I typically aim to prepare well for both, having a short management summary and easily retrievable all supporting evidence.



**Figure 3:** *The Persuading scale: a graphical representation of some Culture Map countries (the cultural context of my recent professional interactions).*

## 16.4: Leading

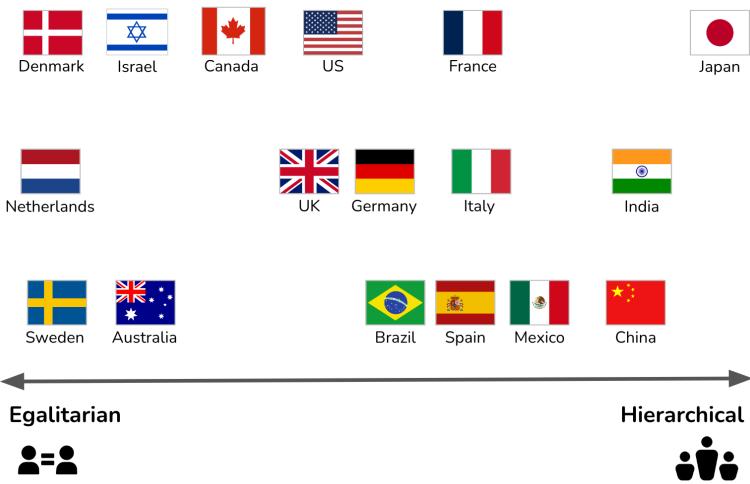
Architects have informal and sometimes formal authority. The leading scale measures the degree of **respect and deference shown to authority figures**.

This scale places countries on a spectrum from **egalitarian to hierarchical**. Egalitarian cultures expect leading to be in a **democratic** fashion. Hierarchical cultures expect leading to be **from top to bottom** (Figure 4).



image by istock

The difference in leadership styles can make an architect's work challenging. The same leadership style can lead different people to perceive an architect as **weak (no leadership)** and **too hard (a dictator)**. The only way to create a working situation is to have an open conversation with the team and agree on expectations and the leadership approach.



**Figure 4:** The Leading scale: a graphical representation of some Culture Map countries (the cultural context of my recent professional interactions).

## 16.5: Deciding

Architectural work is about **making decisions**<sup>2</sup>. The Deciding scale measures the degree to which a culture is **consensus-minded**.



image by istock

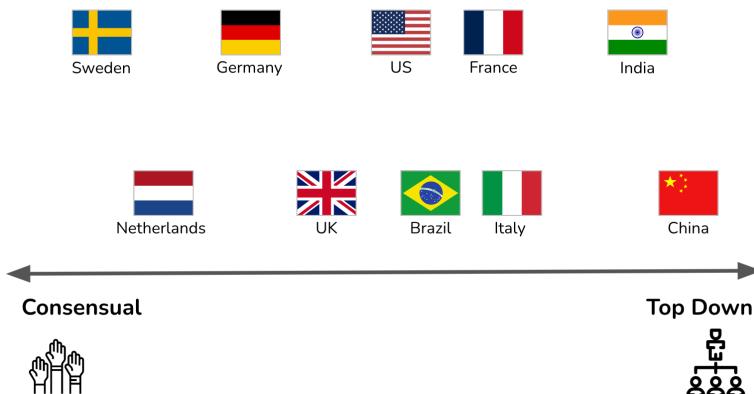
According to Meyers, we often assume that the most egalitarian cultures will be the most democratic, while the most hierarchical ones will allow the boss to make unilateral decisions. But this is only sometimes the case. Germans are more hierarchical than Americans but more likely than their U.S. colleagues to build group agreements before making decisions. The Japanese are both strongly hierarchical and strongly consensus-minded (Figure 5).

Similar to the Leading scale, the difference in deciding styles can make an architect's work complicated. I have been in situations where the different members of the same team have had radically different expectations regarding decision-making: some were sitting and waiting for an architect to come up with a decision, and others were offended by any decision that was not complete consensus. Again, the only way to create a working situation is to have an open conversation with the team and **agree on**

---

<sup>2</sup><https://architectelevator.com/gregors-law/>

**expectations and the decision approach.** One approach I used is a hybrid option: agreeing with a team to try to come up with a decision based on consensus but delegating the decision to an architect when an agreement was impossible.



*Figure 5: The Deciding scale: a graphical representation of some Culture Map countries (the cultural context of my recent professional interactions).*

## 16.6: Trusting

Architects need to build trust with multiple stakeholders. The culture map scale defines two extremes; **task-based cognitive trust** (from the head) and **relationship-based affective trust** (from the heart).

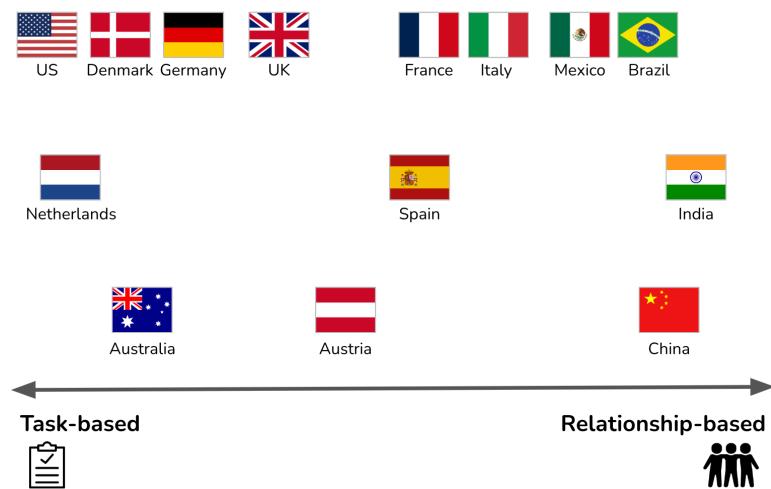


image by istock

In **task-based** cultures, trust is built cognitively **through work**. We feel mutual trust if we collaborate, prove ourselves reliable, and respect one another's contributions.

In a **relationship-based** society, trust results from weaving a solid **affective connection**. We establish trust if we spend time laughing and relaxing together, get to know one another personally, and feel a mutual liking (Figure 6).

Without trust, architects' impact is limited. The best way for architects to build trust is to **align their working methods** with the rituals of the teams they are working with. In particular, finding time to attend events such as all-hands or off-site gatherings of groups and having regular 1:1 meetings with key stakeholders can be an efficient way to gain trust.



**Figure 6:** *The Trusting scale: a graphical representation of some Culture Map countries (the cultural context of my recent professional interactions).*

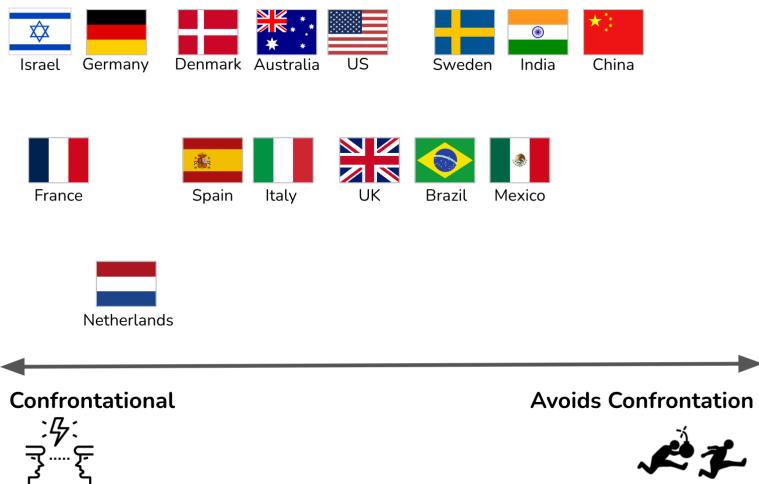
## 16.7: Disagreeing

Architectural work may lead to many disagreements and conflicts. Different cultures have very different ideas about how **productive confrontation** is for a team or an organization. This scale measures **tolerance for open debate** and inclination to see it as helpful or harmful to collegial relationships (Figure 7).



image by istock

Like the Leading and Deciding scales, architects need to have an open conversation with the team and agree on how to disagree. **Disagreeing is an unavoidable** part of the work of architects that want to make an impact. Due to the higher diversity of their audiences, architects must also be extra attentive to the cultural aspects of disagreeing to avoid taking too personally what others consider a routine work discussion.



*Figure 7: The Disagreeing scale: a graphical representation of some Culture Map countries (the cultural context of my recent professional interactions).*

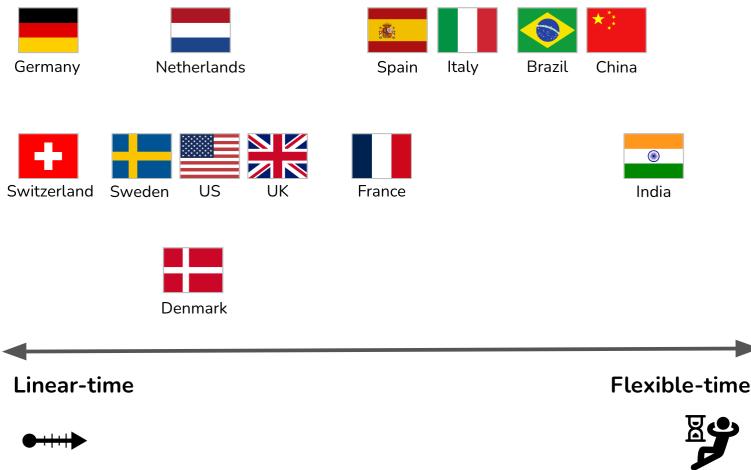
## 16.8: Scheduling

Architects will need to participate in many meetings and projects. All businesses follow agendas and timetables, but in some cultures, people **strictly adhere to the schedule**. In others, they treat it as a **suggestion**. The Scheduling scale assesses how much people value operating in a structured, linear fashion versus being flexible and reactive. This scale is based on the “monochronic” and “polychronic” distinction formalized by Edward Hall (Figure 8).



image by istock

Due to more exposure to diverse audiences, my rule of thumb is that architects should **be on time** according to the more linear interpretation and **tolerate those who are not**. But more importantly, adapt to the overall rhythms.



**Figure 8:** The Scheduling scale: a graphical representation of some Culture Map countries (the cultural context of my recent professional interactions).

## 16.9: Rules

I also found Erin Meyer's four rules on how to bridge the cultural gaps:

- **Rule 1: Don't Underestimate the Challenge.** It's not always easy to bridge cultural gaps. Management styles stem from habits developed over a lifetime, which makes them hard to change.
- **Rule 2: Apply Multiple Perspectives.** Where a culture falls on a scale doesn't in itself mean anything. What matters is the position of one country relative to another.
- **Rule 3: Find the Positive in Other Approaches.** People tend to see the negative when looking at how other cultures work. But if you understand how people from varied backgrounds behave, you can turn differences into the most significant assets.
- **Rule 4: Adjust and Readjust, Your Position.** It's not enough to shift to a new position on a single scale; you'll need to widen your comfort zone to move more fluidly back and forth along all eight.

## 16.10: Questions to Consider

- *How would you describe your communication style based on Erin Meyer's model? Are you more low-context or high-context?*
- *How do you prefer to give and receive feedback? Do you prefer a more direct or indirect approach?*
- *When it comes to persuasion, do you prefer specific cases and examples or more holistic explanations?*
- *How do you see leadership? Do you prefer a hierarchical or egalitarian structure in your work environment?*
- *What's your approach to decision-making? Do you prefer consensus or top-down decisions?*
- *How do you build trust? Do you base it more on personal relationships or work-based achievements?*
- *How do you handle disagreements? Do you prefer to tackle them directly or avoid confrontations?*
- *How do you perceive time and schedule? Do you consider time linear and absolute or a flexible range?*
- *What strategies do you use to adapt to the communication styles of different cultures and professional communities?*
- *How do you adjust your leadership or decision-making approach when dealing with team members from different cultures?*
- *How do you maintain trust in multicultural environments? What challenges have you faced in this regard?*
- *How do you handle disagreements in a multicultural context?*
- *In which areas of Meyer's model could you improve?*
- *How would you handle a situation where different members of the same team have radically different expectations regarding decision-making or disagreeing?*

# 17: Managing Organizational Complexity: Six Simple Rules



image by nat aggiate from pixabay

**IN THIS SECTION, YOU WILL:** Get an introduction to Six Simple Rules, a model for setting up organizational structures based on cooperation.

**KEY POINTS:**

- The Six Simple Rules approach emphasizes that in today's complicated business environment, you must set up organizational structures based on cooperation.
- To deal with complexity, organizations should depend on the judgment of their people and on these people cooperating.
- This view is well aligned with the ideas of Grounded Architecture.

The book **Six Simple Rules: How to Manage Complexity without Getting Complicated**<sup>1</sup>, by Yves Morieux and Peter Tollman, is another source of inspiration for my vision of the Architecture practice. Morieux and Tollman introduced the concept of **Smart Simplicity** with six rules or strategies that enable organizations to promote new behaviors and improve performance.

The Six Simple Rules approach emphasizes that in today's business environment, you **need to set up organizational structures based on cooperation**. To deal with complexity, organizations should depend on the judgment of their people, giving them more autonomy to act. It also depends on these people cooperating to utilize the organization's capabilities to cope with complex problems.

In this chapter, I explore the connection between the ideas of Grounded Architecture and Six Simple Rules. The Six Simple Rules ideas have been a very inspirational source of my work. **Conway Law**<sup>2</sup> illustrates that the link between organizational structures and IT architecture is strong. In addition, the Six Simple Rules approach and architecture work deal with the issue of **managing complexity**.

<sup>1</sup><https://bcg.com/capabilities/organization/smart-simplicity/six-rules-overcoming-complexity>

<sup>2</sup><https://martinfowler.com/bliki/ConwaysLaw.html>

## 17.1: Background: Limitations of Hard and Soft Management Approaches

One of the Six Simple Rules' central premises is that **conventional management approaches**, which the authors split into hard and soft, are neither sufficient nor appropriate for the complexity of organizations nowadays.

The **hard approach** rests on two fundamental assumptions:

- The first is the belief that **structures, processes, and systems** have a direct and predictable effect on performance, and as long as managers pick the right ones, they will get the performance they want.
- The second assumption is that the **human factor is the weakest and least reliable link** of the organization and that it is essential to **control people's behavior through the proliferation of rules** to specify their actions and through financial incentives linked to carefully designed metrics and key performance indicators (KPIs) to motivate them to perform in the way the organization wants them to.

When the company needs to meet new performance requirements, the **hard response** is to **add new structures, processes, and systems** to help satisfy those requirements. Hence, introducing the innovation czar, the risk management team, the compliance unit, the customer-centricity leader, and the cohort of coordinators and interfaces have become so common in companies.

On the other end, we have a soft management approach. According to the **soft approach**, an organization is a set of **interpersonal relationships and the sentiments** that govern them.

- **Good performance is the by-product of good interpersonal relationships.** Personal traits, psychological needs, and mindsets predetermine people's actions.

- To change behavior at work, you need to **change the mindset (or change the people)**.

Both approaches are limited in today's world and are harmful to cooperation. A **hard approach introduces complicated mechanisms**, compliance, and “checking the box” behaviors instead of the engagement and initiative to make things work. The **soft approach’s emphasis on good interpersonal feelings creates cooperation obstacles** as people want to maintain good feelings.

## 17.2: Six Simple Rules Overview

The Six Simple Rules approach covers two areas: **autonomy** and **cooperation**. The first three rules create the conditions for **individual autonomy and empowerment** to improve performance.

- **Understand what your people do.** Trace performance back to behaviors and how they influence overall results. Understand the context of goals, resources, and constraints. Determine how an organization's elements shape goals, resources, and constraints.
- **Reinforce integrators.** Identify integrators—those individuals or units whose influence makes a difference in the work of others—by looking for points of tension where people are doing the hard work of cooperating. Integrators bring others together and drive processes.
- **Increase the total quantity of power.** When creating new roles in the organization, empower them to make decisions without taking power away from others.

The Six Simple Rules' authors emphasize the difference between Autonomy and Self-Sufficiency. **Autonomy** is about fully mobilizing our intelligence and energy to **influence outcomes**, including those we do not entirely control. **Self-sufficiency** is about limiting our efforts only to those **outcomes that we control entirely without depending on others**. Autonomy is essential for coping with complexity; **self-sufficiency is an obstacle** because it **hinders the cooperation** needed to make autonomy effective.

This difference between **Autonomy** and **Self-Sufficiency** leads us to the second set of rules that compels people to confront complexity and use their newfound autonomy to cooperate with others so that **overall performance, not just individual performance**, is radically improved.

- **Increase reciprocity.** Set clear objectives that stimulate mutual interest to cooperate. Make each person's success dependent on the success of others. Eliminate monopolies, reduce resources, and create new networks of interaction.
- **Extend the shadow of the future.** Have people experience the consequences that result from their behavior and decisions. Tighten feedback loops. Shorten the duration of projects. Enable people to see how their success is aided by contributing to the success of others.
- **Reward those who cooperate.** Increase the payoff for all when they cooperate in a beneficial way. Establish penalties for those who fail to cooperate.

## 17.3: Rule 1: Understand What Your People Do

The first rule states that you must genuinely understand performance: **what people do and why they do it**. When you know why people do what they do and how it drives performance, you can define the **minimum sufficient set of interventions with surgical accuracy**.



image by istock

### 17.3.1: General Guidelines

The Six Simple Rules approach states that you can genuinely understand performance by:

- Tracing performance back to behaviors and how they influence and combine to produce overall results.
- Using observation, mapping, measurement, and discussion.

- Understanding the context of goals, resources, and constraints within which the current behaviors constitute rational strategies for people.
- Finding out how your organization's elements (structure, scorecards, systems, incentives, and so on) shape these goals, resources, and constraints.

### 17.3.2: The Role of Architecture Practice

I have found architecture practice can be very helpful in understanding what people really do in organizations in two ways:

- Having a **Data Foundation** with an overview of various data sources can show where activities are happening, visible trends, and how people cooperate. One of the Data Foundation principles, **build maps, not control units**, supports understanding and orientation rather than being a simple metric tool.
- Leveraging the **People Foundation** to connect people and enable them to learn what is happening in different parts of the organization.

## 17.4: Rule 2: Reinforce Integrators

The Six Simple Rules approach emphasizes the importance of reinforcing integrators by looking at those directly involved in the work, giving them power and interest to foster cooperation in dealing with complexity instead of resorting to the paraphernalia of overarching hierarchies, overlays, dedicated interfaces, balanced scorecards, or coordination procedures.



image by robert\_owen\_wahl from pixabay

### 17.4.1: General Guidelines

You can reinforce integrators by:

- Using feelings to identify candidates: emotions provide essential clues for the analysis because they are symptoms rather than causes.
- Finding operational units that can be integrators among peer units because of some particular interest or power.

- **Removing managerial layers which cannot add value** and reinforcing others as integrators by eliminating some rules and relying on observation and judgment rather than metrics whenever cooperation is involved.

#### **17.4.2: The Role of Architecture Practice**

Architecture practice, in my view, should be strongly related to reinforcing integrators:

- Via the **People Foundation**, architecture practice can **help identify integrators** and connect them to leverage their work.
- Furthermore, my view on architects as **superglue** defines architects as **critical integrators** and **integrator role models** in an organization.
- And having a **Data Foundation** can **support integrators with data and insights**, empowering them to do better, more informed work.

## 17.5: Rule 3: Increase the Total Quantity of Power

Whenever you consider an **addition** to your organization's **structure, processes, and systems**, think about **increasing the quantity of power**. Doing so may **save you from increasing complicatedness** and enable you to achieve a more significant impact with less cost. You can increase the quantity of power by allowing some functions to influence performance and stakes that matter to others.



image by istock

### 17.5.1: General Guidelines

To increase the quantity of power, the Six Simple Rules approach recommends the following actions:

- Whenever you are going to make a design decision that will **swing the pendulum—between center and units, between**

functions and line managers, and so on—see if making some parts of the organization **benefit from new power bases** could satisfy more requirements in dealing with complexity so that you don't have to swing the pendulum in the other direction in the future (which would only compound complicatedness with the mechanical frictions and disruptions inherent to these changes).

- When you have to create new functions, make sure you give them the power to play their role and that this **power does not come at the expense of the power needed by others to play theirs**.
- When you **create new tools** for managers (planning, or evaluation systems, for instance), ask yourself if these constitute **resources or constraints**. Providing a few tools simultaneously is more effective (because it creates a critical mass of power) than many tools sequentially, one after the other.
- **Regularly enrich power bases** to ensure agility, flexibility, and adaptiveness.

### 17.5.2: The Role of Architecture Practice

Architecture practice can support increasing power quantity with the **operating model** that promotes distributing decision-making:

- Via the **People Foundation** you can increase the quantity of the **decision-making power** and keep architectural decision-making **distributed across the organization** and embedded in the development teams. Development teams traditionally have the best insights and most information relevant for making a decision.
- Additionally, the **Data Foundation**, accessible to all interested people in the organization, can give them **data in insights** that can increase their power in daily work.

## 17.6: Rule 4: Increase Reciprocity

In the face of business complexity, work is becoming more interdependent. To meet multiple and often contradictory performance requirements, **people must rely more on each other**. They need to **cooperate directly** instead of depending on dedicated interfaces, coordination structures, or procedures that only add to complicatedness.



image by istock

### 17.6.1: General Guidelines

Reciprocity is the recognition by people or units in an organization that they **have a mutual interest in cooperation** and that the success of one depends on the success of others (and vice versa). The way to create that reciprocity is by setting rich objectives and reinforcing them by:

- **eliminating monopolies,**

- **reducing resources**, and
- **creating new networks of interaction**.

### 17.6.2: The Role of Architecture Practice

Architecture practice can be directly related to increasing reciprocity:

- The **People Foundation** directly supports one of the ways of reinforcing reciprocity: **creating new networks of interactions**.
- The hybrid operating model makes the success of architecture practice dependent on **architects' impact**. Likewise, the developer team's support depends on architecture support. Integrating feedback from groups that architects support in architects' performance evaluations is also crucial for increasing reciprocity between architecture and other units.

## 17.7: Rule 5: Extend the Shadow of the Future

The Six Simple Rules approach emphasizes the importance of making visible and clear **what happens tomorrow as a consequence of what they do today**. You can manage complex requirements by making simple changes while removing organizational complexity. With the strategic alignment typical of the hard approach, these simple solutions—for instance, career paths—often come at the end of a sequence that starts by installing the most cumbersome changes: new structure, processes, systems, metrics, etc. Simple and effective solutions are then impossible.



image by joe from pixabay

### 17.7.1: General Guidelines

The Six Simple Rules approach identifies four ways to extend the shadow of the future:

- Tighten the feedback loop by making **more frequent the moments when people experience the consequence** of the fit between their contributions.
- **Bring the endpoint forward**, notably by shortening the duration of projects.
- **Tie futures together** so that successful moves are conditioned by contributing to the successful move of others.
- Make people **walk in the shoes they make** for others.

### 17.7.2: The Role of Architecture Practice

Architecture practice can extend the shadow of the future in multiple ways:

- The **Data Foundation** can create transparency and provides data necessary to **model the future**. I've used such data to create many **simulations and roadmap options**.
- The principle of applying **economic modeling** to architecture decision-making directly supports describing what happens tomorrow as a consequence of what they do today.

## 17.8: Rule 6: Reward Those Who Cooperate

Lastly, the Six Simple Rules approach recommends that when you cannot create direct feedback loops embedded in people's tasks, you need **management's intervention to close the loop**. Managers must then use the familiar performance evaluation tool but in a very different way.



image by stocksnap from pixabay

### 17.8.1: General Guidelines

To reward those who cooperate, managers:

- Must go beyond technical criteria (putting the blame where the root cause problem originated). In dealing with the business complexity of multiple and often conflicting performance requirements, the smart organization accepts that

problems in execution happen for many reasons and that the only way to solve them is to **reduce the payoff for all those people or units that fail to cooperate in solving a problem**, even if the problem does not take place precisely in their area, and to **increase the payoff for all when units cooperate in a beneficial way**.

- They must not blame failure, but **blame failing to help or ask for help**.
- Instead of the elusive sophistication of balanced scorecards and other counterproductive cumbersome systems and procedures, they can **use simple questions** to change the terms of the managerial conversation so that **transparency and ambitious targets become resources rather than constraints** for the individual. Managers then act as integrators by obtaining from others the cooperation that will leverage the rich information allowed by this transparency and help achieve superior results.

### 17.8.2: The Role of Architecture Practice

Architecture practice can reward cooperation by making it easier for everyone to **help others and ask for help**.

- Having a strong **People Foundation** can provide the context and **networks of people** to collaborate more easily.
- Adding diverse data sources to the **Data Foundation** can create **transparency about cooperation** opportunities and challenges.

## 17.9: Questions to Consider

- How can the concept of Smart Simplicity apply to your current role or position within your organization?
- Do you feel the structures, processes, and systems directly and predictably affect performance in your organization?
- Do you feel that your organization views the human factor is viewed as the weakest link? How does this affect how you and your colleagues perform?
- How do you perceive the balance between your organization's hard and soft management approaches? Is one approach more dominant?
- How does your organization currently promote autonomy and cooperation among employees? Are there areas for improvement?
- How do the assumptions of hard and soft management approaches hinder cooperation in your organization?
- How can you increase the total power within your organization without taking power away from others?
- How can your organization increase reciprocity and make each person's success dependent on the success of others?
- How can your organization extend the shadow of the future? Are there feedback mechanisms in place to make people accountable for their decisions?
- How are those who cooperate rewarded in your organization? Are there mechanisms in place to increase the payoff for all when they cooperate beneficially?
- How can the architecture practice in your organization support the implementation of the Six Simple Rules?
- How do your organization's current systems and structures promote or hinder the cooperation needed to make autonomy effective?

# 18: Understanding Product Development

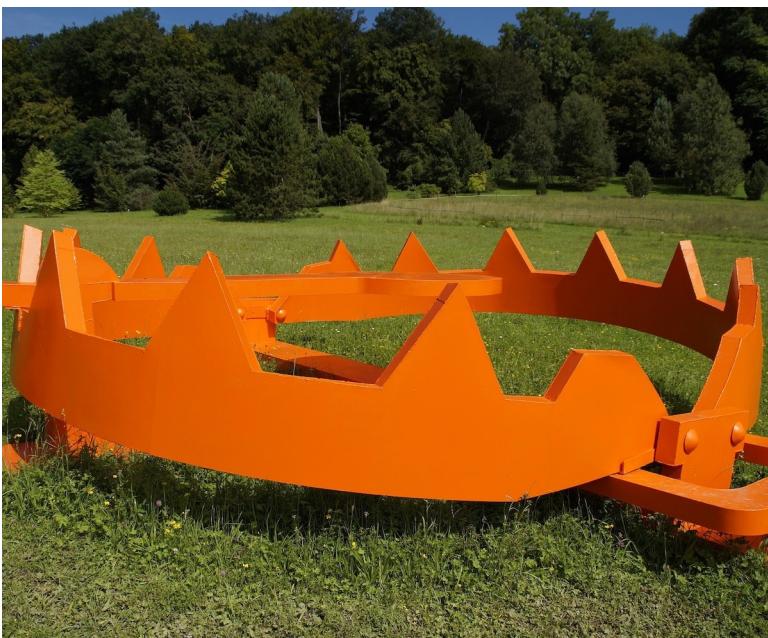


image by m w from pixabay

**IN THIS SECTION, YOU WILL:** Understand the importance of architecture in helping companies become successful product-led organizations, focusing strongly on their customers' actual needs and preferences.

**KEY POINTS:**

- When it comes to product development, I generally recommend two resources for architects: “Escaping the Build Trap: How Effective Product Management Creates Real Value” by Melissa Perri and “The Discipline of Market Leader” by Michael Treacy and Fred Wiersema.
- The build trap occurs when businesses focus too much on their product’s features and functionalities, overlooking customers’ needs and preferences.
- The Discipline of Market Leader highlights three strategic paths a company can use to achieve market leadership: operational excellence, product leadership, and customer intimacy.

Product development is the discipline of creating and bringing **new products or services to the market**. Having a strong product development system is essential for modern organizations to thrive in a competitive, fast-paced, and ever-changing business landscape. It can drive growth, satisfy customer demands, enhance operational efficiency, and build a sustainable brand.

Understanding product development is essential for architects. Product development involves the journey **from the conception** of an idea to the product’s **final development**, marketing, and distribution. Product development encompasses various activities and stages to transform an initial concept into a tangible and market-ready offering, and architects should be involved in these activities.

When it comes to understanding product development, I generally recommend two resources for architects:

- “[Escaping the Build Trap: How Effective Product Manage-](#)

ment Creates Real Value<sup>1</sup>" by Melissa Perri, and

- "The Discipline of Market Leader<sup>2</sup>" by Michael Treacy and Fred Wiersema.

Both of these sources provide several things for architects:

- Increase their awareness about how good or bad product development looks like,
- Provide them with tools to support or challenge product decisions, and
- Prepare them for designing the organization's systems that suit diverse product strategies.
- Enable them to collaborate effectively with product teams (product managers, product operations).

---

<sup>1</sup><https://www.goodreads.com/book/show/42611483-escaping-the-build-trap>

<sup>2</sup><https://hbr.org/1993/01/customer-intimacy-and-other-value-disciplines>

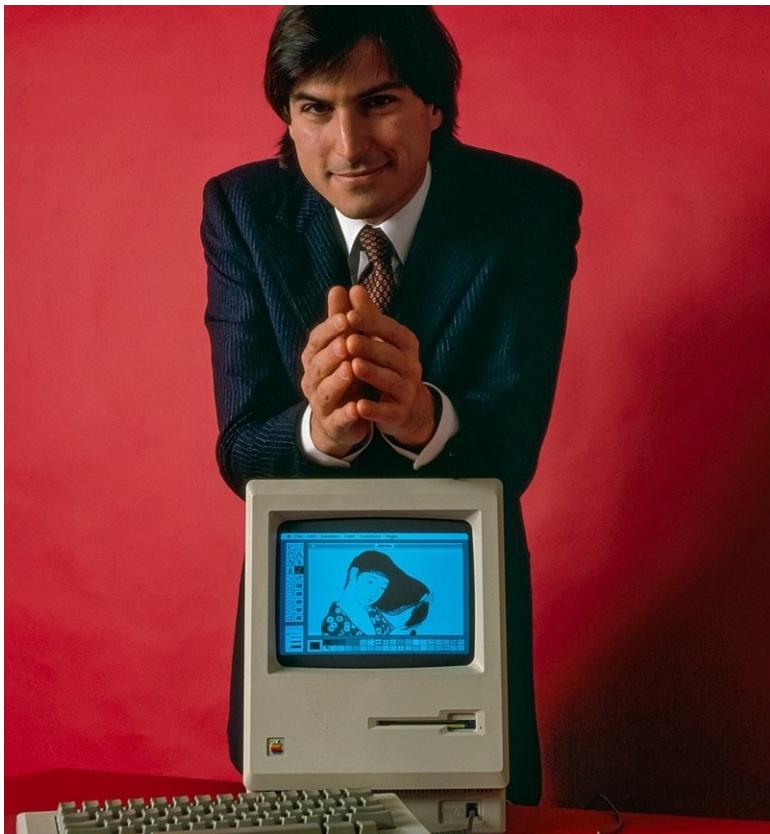


image by wikimedia commons

## 18.1: The Build Trap

The Escaping the Build Trap book is a guide intended to help organizations **shift their focus** from simply **building and shipping** products to **creating value** for their customers. The “build trap” refers to the common pitfall where companies become fixated on building more features and products without considering whether they meet customer needs or generate desired outcomes.

Perri explores the reasons behind the build trap and provides practical strategies to help businesses escape it by adopting a **value-centric, customer-focused approach**. The main message is that by understanding customer needs, adopting a customer-centric approach, and **embracing innovation and agility**, companies can increase their chances of developing successful products that stand out in the market.

In addition to numerous insights for architects, the book provides the following valuable tips necessary for architects’ good interaction with product development:

- Know how a **good product development** process looks like
- Recognizing **bad product-development** approaches
- Identifying **bad product manager** archetypes

### 18.1.1: How a Good Product Development Approach Looks Like

Product-led companies understand that the success of their products is the primary **driver of growth and value for their company**. They prioritize, organize, and strategize around product success.

Critical elements of successful product-led companies include:

1. **Understanding Customer Needs:** Successful companies understand customer needs, desires, and expectations to develop successful products. Businesses can gain valuable insights and tailor their products by conducting thorough market research and engaging with customers.
2. **Adopting a Customer-Centric Approach:** Organizations need to adopt a customer-centric approach to product development to avoid the build trap. This approach means prioritizing customer satisfaction and incorporating their feedback throughout the entire product development process.
3. **Executing Iterative Product Development:** Iterative product development is essential, continuously testing and refining the product based on customer feedback. An iterative process helps businesses identify and address potential issues before they become significant problems.
4. **Aligning Business Goals with Customer Needs:** Businesses should align their goals and objectives with the needs of their customers. Doing so can ensure their products deliver value and create a robust and loyal customer base.
5. **Embracing Innovation and Agility:** Businesses must be innovative and agile to adapt to rapidly changing customer preferences and market conditions. This adaptivity includes staying informed about the latest trends, technologies, and best practices in product development.
6. **Measuring Success:** Accurately measuring a product's success is essential. Such measuring involves tracking key performance indicators (KPIs) and using data-driven insights to make informed product improvements and enhancements decisions (Figure 1).

Architects should be familiar with these characteristics, helping their product leads to operate an effective product development.

	Category	Metric
	<b>Acquisition</b> measure when someone first starts using your product or service	Number of new signups or qualified leads
		Customer acquisition cost (CAC)
	<b>Activation</b> show how well you are moving users from acquisition to moment where they discover why your product is valuable to them and, in turn, provide value to your business	Activation rate
		Time to activate
		Free-to-paid conversions
	<b>Engagement</b> measure how (and how often) users interact with your product	Monthly, weekly, daily active users (MAU, WAU, DAU)
		Stickiness (DAU/MAU)
		Feature usage
	<b>Retention</b> gauge how many of your users return to your product over a certain period of time	Retention rate
		Churn rate
		Customer lifetime value (CLV)
	<b>Monetization</b> capture how well your business is turning engagement into revenue	Net revenue retention (NRR)
		Monthly recurring revenue (MRR)
		Average revenue per user (ARPU)

*Figure 1: Some of the typical product metrics.*

### 18.1.2: Bad Product Companies Archetypes

The build trap occurs when businesses focus too much on their product's features and functionalities, overlooking customers' needs and preferences. Value, from a business perspective, is pretty straightforward. It can fuel your business: **money, data, knowledge capital, or promotion**. Every feature you build, and any initiative you take as a company should result in some outcome that is tied back to that business value.

Many companies are, instead, led by sales, visionaries, or technology. All of these ways of organizing can land you in the build trap.

1. **Sales-led companies** let their contracts define their product strategy. The product roadmap and direction were driven by

what was promised to customers without aligning with the overall strategy.

2. **Visionary-led companies** can be compelling — when you have the right visionary. Also, when that visionary leaves, the product direction usually crumbles. Operating as a visionary-led company is not sustainable.
3. **The technology-led companies** are driven by the latest and coolest technology. The problem is that they often lack a market-facing, value-led strategy.

Architects should be able to recognize and frequently challenge organizations with these archetypes.

### 18.1.3: Bad Product Manager Archetypes

Architects will frequently need to collaborate closely with product managers. The fundamental role of the product manager in the organization is to work with a team to create the right product that **balances meeting business needs with solving user problems**. Product managers connect the dots. They take input from customer research, expert information, market research, business direction, experiment results, and data analysis.

To better understand the role of a product manager, it is helpful to understand three bad product manager archetypes:

- The Mini-CEO,
- The Former Project Manager, and
- The Waiter.

### 18.1.3.1: The Mini-CEO

Product managers are not the mini-CEOs of a product, yet, according to Melissa Perry, **most job postings for product managers** describe them as the mini-CEO.

CEOs have sole authority over many things. Product managers can't change many things a CEO can do in an organization. They especially **don't have power over people** because they are not people managers at the team level.

Instead, they must influence them to move in a specific direction. Out of this CEO myth emerged an archetype of a very **arrogant product manager** who thinks they rule the world.



image by istock

### 18.1.3.2: The Former Project Manager

Product managers are not project managers, although some project management is needed to execute the role correctly.

**Project managers are responsible for the when.** When will a project finish? Is everyone on track? Will we hit our deadline?

**Product managers are responsible for the why.** Why are we building this? How does it deliver value to our customers? How does it help meet the goals of the business? The latter questions are more challenging to answer than the former, and product managers who don't understand their roles often resort to doing that type of work.

Many companies still think the project manager and product manager are the same.

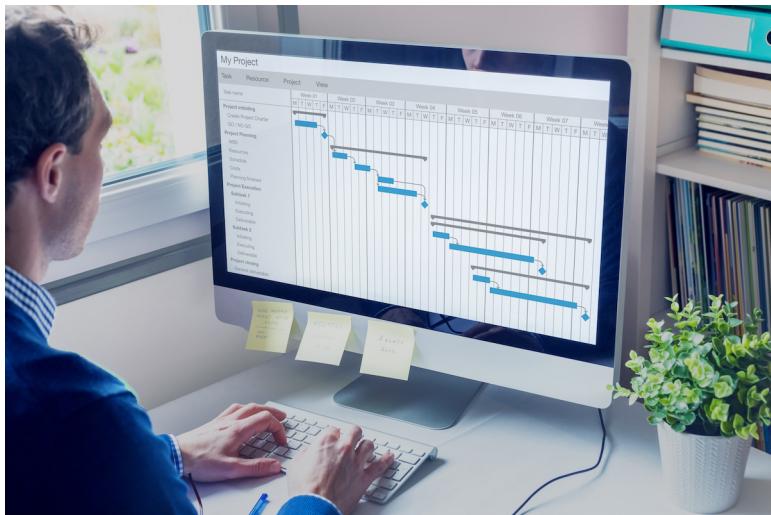


image by istock

### 18.1.3.3: The Waiter

The waiter is a product manager who, at heart, is an **order taker**. They go to their stakeholders, customers, or managers, ask for what they want, and turn those desires into a list of items to be developed. There is **no goal, vision, or decision-making** involved. More often than not, the most important person gets their features prioritized.

Instead of discovering problems, waiters ask, “What do you want?” The customer asks for a specific solution, which these product managers implement.

The waiter approach leads to what David J. Bland is calling the **Product Death Cycle**<sup>3</sup>:

- No one uses our product,
- Ask customers what features are missing,
- Build the missing features (which no one uses, starting the cycle again).



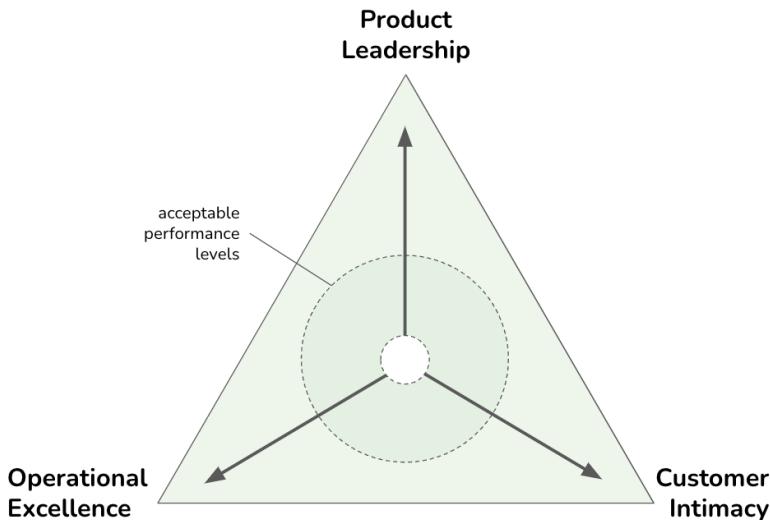
image by istock

---

<sup>3</sup><https://twitter.com/davidjbland/status/467096015318036480>

## 18.2: The Discipline of Market Leader

Another tool I found helpful in my work as an architect is **The Discipline of Market Leader**<sup>4</sup>, a concept developed by Michael Treacy and Fred Wiersema. The Discipline of Market Leader highlights a company's three strategic paths to achieve market leadership: **operational excellence**, **product leadership**, and **customer intimacy** (Figure 2).



**Figure 2:** The Discipline of Market Leader model postulates that any successful business needs to maintain at least “acceptable” levels of performance in each of the three dimensions (operational excellence, product leadership, and customer intimacy) but would need to choose one of them to become a market leader in its field.

**Product Leadership** companies provide leading-edge products or useful new applications of existing products or services. Their core process includes invention, commercialization, market exploita-

---

<sup>4</sup><https://hbr.org/1993/01/customer-intimacy-and-other-value-disciplines>

tion, and disjoint work procedures. Exemplars are Apple, Tesla, Nike, Rolex, and Harley-Davidson.

**Operational Excellence** companies provide reliable products and services at competitive prices, delivered with minimal difficulty or inconvenience. Their value proposition is **guaranteed low price and hassle-free service**. Which also includes time spent to purchase, future product maintenance, and ease of getting swift and dependable service. The core processes include product delivery, basic service cycle + build on standards, no frills fixed assets. Exemplars are IKEA, McDonald's, Starbucks, Walmart, and Southwest Airlines.

**Customer Intimacy** companies do not deliver what the market wants but **what a specific customer wants**. Creating results for carefully selected and nurtured clients. Continually tailors products/services to customers to offer the '**best total solution**'. Exemplars are Salesforce, LMS Providers, HomeDepot.

The model postulates that any successful business needs to maintain at least "acceptable" performance levels in each of the three dimensions but would need to **choose one of them to become a market leader** in its field. The model suggests that if you genuinely want to excel in any of the three disciplines, you must make **sacrifices in the other two** as these become mutually exclusive. By focusing on one (and one only) value to excel at, they beat competitors by dividing their attention and resources among more than one discipline. Each value discipline demands a **different operating model to capture the value best**. And customers know that to expect superior value in every dimension is unreasonable. You don't go to Walmart for the best personalized service, or buy Nike sneakers because of low prices.

I found the model helpful in two ways:

- To challenge **plans and strategies that are too ambitious**, e.g., wanting to excel in all three directions: operational

excellence, product leadership, and customer intimacy. It is very complex and expensive to try to build a scalable solution accessible by many users and from many countries while addressing the particular needs of one local customer while doing significant work to innovate around the latest technology hype.

- To prepare for architecting the company's IT landscape, as different directions require different approaches.

Each direction represents a unique value proposition and requires specific architectural considerations to support it effectively.

- **Operational Excellence:** This path focuses on delivering products or services at the lowest cost and highest efficiency. The IT architecture should **streamline processes, automate repetitive tasks, and optimize resource allocation**. It involves leveraging technologies like enterprise resource planning (ERP) systems, supply chain management tools, and process automation solutions to achieve operational efficiency. **Scalability, reliability, and cost-effectiveness** are critical factors in architectural design.
- **Product Leadership:** This path involves developing and delivering innovative and superior products or services that differentiate an organization from its competitors. The IT architecture should prioritize **flexibility, agility, and the ability to support rapid innovation**. Architecture typically emphasizes integrating product development and research systems, **data analytics** capabilities, and **collaboration tools** to facilitate idea generation, prototyping, and testing. Ensure that the architecture enables seamless integration with external partners and suppliers to foster a culture of innovation and continuous improvement.
- **Customer Intimacy:** This path focuses on building **strong customer relationships** and delivering **personalized experiences**. The IT architecture should enable customer data

collection, analysis, and utilization to gain insights and provide customized solutions. Architectural considerations frequently include complex customer relationship management (CRM) systems, data analytics platforms, and customer engagement tools. Integration with various touchpoints, such as web portals, mobile applications, and social media channels, is essential to deliver a seamless and personalized customer experience.

Incorporating the Discipline of Market Leader into IT architecture design requires **aligning technology choices and design decisions with the chosen strategic path**. Additionally, the architecture should be flexible enough to adapt to evolving market dynamics and business needs, allowing the organization to switch or combine strategic routes if required.

It is important to remember that the Discipline of Market Leader is **not a one-size-fits-all** approach, and organizations may need to balance elements from multiple paths based on their specific business context and market conditions.

## 18.3: Product Operations

Another product concept relevant to architectural practice is **Product Operations**. Melissa Perri and Denise Tilles provide an excellent overview of Product Operations in their book<sup>5</sup>. Perri and Tilles define Product Operations as the discipline of helping your product management function scale well, surrounding teams with all of the essential inputs to set strategy, prioritize, and streamline ways of working.



image by gerd altmann from pixabay

Perri and Tilles define Product Operations structure as consisting of three pillars:

1. **Business Data and Insights:** This pillar focuses on the internal collection and analysis of data to create and monitor strategies. It helps leaders track progress regarding outcomes, reconciling research and development (R&D) spending with

---

<sup>5</sup><https://www.productoperations.com/>

return on investment (ROI). It also integrates business metrics such as annual recurring revenue (ARR) and retention rates with product metrics, aiding strategic decisions for leaders and product managers.

2. **Customer and Market Insights:** Unlike the first pillar, which deals with internal data, this one aggregates and facilitates research received externally. It includes streamlining insights from customers and users and making them readily accessible for team exploration. Additionally, it provides tools for market research, such as competitor analysis and calculations of total addressable market/serviceable addressable market (TAM/SAM) for potential product ideas.
3. **Process and Practices:** This pillar enhances the value of product management through consistent cross-functional practices and frameworks. It defines the company's product operating model, outlining how strategy is created and deployed, how cross-functional teams collaborate, and how the product management team functions. This area also includes product governance and tool management.

Product Operations has risen as an approach to supporting product and development teams when they need more structured systems for **managing workflows and communications, avoiding chaos, wasted efforts, interdepartmental tensions**, and creating products that fail to meet market needs.

### **18.3.1: Discovery and Ideation Processes**

Product Operations can play an essential role in defining **robust discovery and ideation processes** in product development to ensure that products meet actual market needs. Frequently, products are developed based on assumptions without sufficient market research or user validation. The lack of such processes can lead to products that do not resonate with users and are **misaligned with customer expectations**.

Product Operations aim to facilitate deep discovery work through **structured research sprints** involving surveys, interviews, and direct interactions with user environments. These efforts help identify key pain points and opportunities for innovation. Following the discovery phase, **ideation workshops** allow for the generation of potential features evaluated based on criteria like customer value, feasibility, and alignment with the product vision.

Product Operations frequently work on implementing a **priority matrix** and revamping the **product roadmap** to ensure that the organization directs engineering efforts toward the most validated and impactful opportunities.

Product Operations also formalize **continuous learning and feedback mechanisms** to maintain alignment with evolving **customer needs and market dynamics**.

### **18.3.2: Aligned Data-Driven Planning Processes**

Product Operations typically facilitate the alignment of organizational goals with team autonomy by introducing **regular strategic planning sessions**, such as quarterly roadmap summits. These summits involve key stakeholders and aim to **balance discovery insights with available engineering resources, operational support, and market needs**. Features are evaluated and prioritized based on customer value, development cost, and overall coherence with the platform strategy.

An essential component of this approach is the **continuous collection and analysis of user feedback**, alongside regular review of usage metrics. This data-driven strategy allows teams to adapt quickly if features do not meet performance expectations or user satisfaction, thus **continually refining the product-market fit**.

### 18.3.3: User and Market Feedback Loops

Product Operations can also help organizations is in maintaining and enhancing **product-market fit** through continuous optimization cycles post-market release.

A key component of continuous optimization is the **establishment of robust feedback loops**. These loops, involving **surveys, interviews, and direct observation**, are essential for staying in tune with evolving customer needs and pain points. Product Operations should also ensure that there are reliable systems in place for **showcasing functionality still under development**, allowing for user input well before final releases.

Overall, by stewarding communication, documentation, and insights across teams and stakeholders, Product Operations can foster a **culture of continuous learning and adaptation**. Such an approach helps avoid insular planning and aligns product development more closely with actual user needs and market dynamics, reducing waste and ensuring that **investments are validated** before being fully committed.

### 18.3.4: Product Operations and Architecture Practice

In many ways, the concept of Product Operations resonates with my view of architectural practice. The main difference is that Product Operations maintain a closer relationship with customers, designers, and researchers, bringing end-user perspective much more prominently into focus.

In organizations with Product Operations teams, architecture practice can **create powerful synergic relationships**. Like architecture practice, Product Operations aim to maintain **efficiency and cohesion across departments**. Product Operations function like

an **orchestra conductor** by effectively managing critical data, activities, and communications, ensuring that each team contributes optimally to a unified vision.

In my experience, **Product Operations can make architecture practice more effective** by providing additional insights and data and making it easier for architects to be present at critical moments and interact with key stakeholders. Architecture practice can help Product Operations with complementary insights, data, and stakeholder connections.

## 18.4: Questions to Consider

- *Have you ever found yourself or your organization falling into the “build trap”? What were the signs?*
- *Reflecting on your organization, would you say it’s sales-led, visionary-led, or technology-led?*
- *Can you identify instances where a product manager has acted like a “Mini-CEO,” “Waiter,” or a “Former Project Manager”? What were the consequences?*
- *How does your organization currently understand and incorporate customer needs? Could there be improvements in this area?*
- *How does your company approach iterative product development?*
- *Are your business goals aligned with customer needs? How do you maintain this alignment as business goals and customer needs evolve?*
- *How innovative and agile do you consider your organization to be? What areas need more flexibility or creativity?*
- *What metrics does your organization use to measure product success?*
- *Which of the three strategic paths (operational excellence, product leadership, and customer intimacy) does your company or project currently emphasize most? Why?*
- *Can you identify areas where your company or project may be trying to excel in all three disciplines, potentially causing complexity or inefficiency?*
- *How does your IT architecture support your company’s strategic path? Are there areas where it could better align?*
- *How can incorporating the Discipline of Market Leader into your IT architecture design influence your technology choices and design decisions?*

# **19: Architecture Governance: Nudge, Taxation, Mandates**



image by nonbirinonko from pixabay

**IN THIS SECTION, YOU WILL:** Understand that a technology governance model should be a well-balanced hybrid of three different styles of governing: mandates and bans, taxes, and nudging.

**KEY POINTS:**

- Architecture practice should support governance models adaptable to organizations' complex and diverse needs. A technology governance model should be a well-balanced hybrid of three different styles of governing: mandates and bans, taxes, and nudging.
- Nudging is a form of governing where you create subtle or indirect suggestions influencing someone's behavior or decision-making without forcing them or limiting their freedom of choice.
- Governing with taxes (economic incentives) is a form of guiding in which people are not forbidden to make some decisions but need to "pay" some form of taxes on used resources.
- With mandates and bans, you guide people by explicitly defining what they should or should not do.

Governance refers to the framework of rules, practices, and processes by which an organization is directed and controlled. It encompasses the mechanisms by which an organization's goals are set, pursued, and monitored, ensuring accountability, fairness, and transparency. Governance can be applied to various domains, including corporate, IT, project, and data governance. **IT architecture is a form of governance** because it establishes structured frameworks for managing and controlling an organization's technology resources and processes. It ensures alignment with business objectives, promotes standardization, manages risks, optimizes resources, facilitates change management, supports decision-making, measures performance, and fosters innovation.

The difficulty of governance stems from the need to navigate a complex web of diverse interests, rapidly changing conditions, and multifaceted challenges. There is no one-fit-all form of gover-

nance. Effective governance requires adaptability, collaboration, and a commitment to addressing immediate and long-term issues.

Architecture practice should support governance models that are aligned and adaptable to organizations' complex and diverse needs. Consequently, I see an architecture governance model as a well-balanced hybrid of three different styles of governing:

- **nudging,**
- **taxes** (economic incentives), and
- **mandates and bans.**

## 19.1: Nudging

In behavioral economics and psychology, a **nudge** is a subtle or indirect suggestion influencing someone's behavior or decision-making without forcing them or limiting their freedom of choice. Nudges can be applied in various settings, such as policy-making, marketing, and personal interactions, to encourage people to make better choices, improve their well-being, or achieve specific goals.

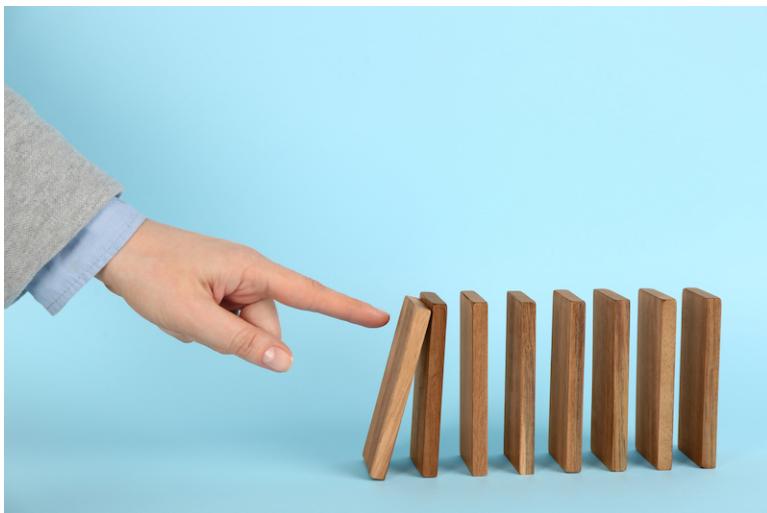


image by istock

A nudge can take many forms, such as a slight change in the environment, a gentle reminder, a positive reinforcement, or a default option. For example, placing healthy food options at eye level in a cafeteria can nudge people to choose healthier meals. Or setting a default option for organ donation can increase the number of donors.

The concept of a nudge was popularized by the book “Nudge: Improving Decisions About Health, Wealth, and Happiness” by Richard Thaler and Cass Sunstein, which argues that various

cognitive biases and heuristics often influence people's decisions, and that nudges can help people overcome these biases and **make better choices**.

Richard Thaler and Cass Sunstein also introduced the concept of **choice architecture** as a critical component of nudging. It refers to how the options are presented to individuals, which can significantly influence their choices. Choice architecture is **the design of the decision-making environment**, which includes the layout, structure, and organization of available options.

In IT architecture, examples of nudging include:

- Architectural **principles** as informal decision guidelines. Such principles do not prescribe a solution, but can subtly guide alignment.
- Recommendations for **best practices** to stimulate introduction and alignment around such practices,
- Default options for technology choices via **golden paths**<sup>1</sup>
- **Highlighting** bad quality software on a **Data Foundation** dashboards to create subtle pressure for people to improve it,
- Tracking of **tech debt** to create awareness about its size and lead action to reduce it,
- **Visualizing cost trends** of cloud services per team to stimulate teams to improve the performance efficiency of their software.

Nudges can frequently lead to better alignment and more harmonization without the negative consequences of mandates, bans, or taxation.

Grounded Architecture is well aligned with ideas of nudging. I designed many **Data Foundation** tools to **highlight areas and issues**

---

<sup>1</sup><https://engineering.spotify.com/2020/08/how-we-use-golden-paths-to-solve-fragmentation-in-our-software-ecosystem/>

we wanted (nudged) people to improve. And the [People Foundation](#) can create mechanisms for sharing experiences, promoting **positive examples**, and capturing lessons learned to help people to make better, more informed decisions. And in the [Architecture Activities Platform](#), I use the operating model that stimulates people to make decisions autonomously but **nudges them to stay well-aligned** and connected to the organizational strategic direction.

## 19.2: Taxation (Economic Incentives)

Governing with taxes is a form of guidance in which we do not forbid people to make choices or decisions, but they **need to “pay” some form of taxes on used resources**. For instance, costs of public cloud usage could be **cross-charged across the organization**, providing a helpful feedback loop to optimize our systems and avoid unnecessary resource consumption, avoiding the “tragedy of commons” which frequently happens when there are no limits on shared resource consumption (e.g., public cloud budget). Compared to nudging, taxes are not only an information feedback loop, but they have consequences (e.g., some projects could be banned if they exceed the cross-charged IT budget).



image by steve buissinne from pixabay

The role of architecture practice in this form of governance should be to ensure that “taxation” is **data-driven and transparent** and to create efficient feedback loops on critical metrics related to “taxes.”

The **Data Foundation** can include and provide all **data regarding “taxes,”** for instance, via insights based on public cloud cost reports. The **People Foundation** can facilitate **aligning processes, goals,**

and working methods to ensure that taxation leads to desired and **meaningful change**.

## 19.3: Mandates and Bans

By governing with mandates and bans, I mean guiding people by **explicitly defining what they should or should not do**. In places I worked in, such mandates and bans have had a limited but important place to **define broader strategic boundaries of choices** people can make. For instance, restricting the usage of public cloud providers to specific vendors or following **strict privacy and security procedures** needs to be explicitly defined and controlled.

You should **use bans with care** and as a last resort to avoid unnecessary blocking or slowing down development and innovation. Nevertheless, they could help clarify critical topics where nudging or taxation would not be sufficient. For instance, having clear rules and control mechanisms to avoid breaking privacy or financial laws can prevent unnecessary incidents and damage.



image by tumisu from pixabay

The role of architecture in this form of governing should be to

**be a stakeholder but not a sole owner in defining mandates and bans.** Mandates and bans frequently need to be defined in collaboration with others, such as security and legal functions. The architecture practice can help by **creating clarity and providing transparency.**

The **Data Foundation** is crucial in creating **clarity and transparency**, for instance, via insights security reports or maps of areas in source code or infrastructure that needed monitoring and controlling based on privacy or security requirements.

The **People Foundation** can help propagate the decision and ensure its **positive impact and acceptance**. You should never order or forbid people to do some things routinely. Spending enough time with all stakeholders to explain the **reasons and motivations** behind introducing some limitations is crucial to ensure the effective working of mandates and bans. A strong People Foundation provides strong connections with key stakeholders and can leverage them to change ways of working more smoothly.

## 19.4: Questions to Consider

- *What are the key components of the governance model in your organization, and how do mandates, taxes, and nudging influence them?*
- *How does your organization currently handle mandates and bans? Are they explicit and aligned with the overall technology strategy?*
- *How effective is the enforcement of these mandates and bans in your organization? Could improvements be made in creating clarity and providing transparency?*
- *How does your organization approach taxation as a form of governance? Is it transparent, data-driven, and efficient?*
- *Can you identify any examples of ‘nudging’ in your current architectural environment? How effective are these subtle suggestions in influencing behavior or decision-making?*
- *How does your organization promote best practices and align around them? Are there any ‘golden paths’ for technology choices?*
- *How are your organization’s tech debt and the cost trends of cloud services tracked and visualized? Do these methods create enough awareness to stimulate improvement?*
- *How could you better utilize nudging to improve organizational decision-making? What biases or barriers to effective decision-making could you target with this approach?*

# **20: Economic Modeling: ROI and Financial Options**



image by nattanan kanchanaprat from pixabay

**IN THIS SECTION, YOU WILL:** Get two answers to the question of the economic value of architecture: the return on investment metaphor and the selling options metaphor.

**KEY POINTS:**

- Architects are frequently asked about the (economic) value of architecture or technology investments.
- Answering this question is a crucial skill for any senior architect. But it may be difficult to answer this seemingly harmless question concisely and convincingly to a non-technical audience.
- Borrowing from existing literature, I sketch two answers to the question of the economic value of architecture: the return on investment metaphor and the selling options metaphor.

Economic and risk modeling is an essential exercise in organizations. Organizations conduct financial and risk modeling exercises, such as ROI calculations, for several key reasons:

- **Decision-Making Support:** Evaluate investments and compare alternatives to allocate resources effectively.
- **Risk Management:** Identify potential risks and perform sensitivity analysis to anticipate and mitigate issues.
- **Budgeting and Planning:** Aid in resource allocation, detailed budgeting, and long-term forecasting.
- **Performance Measurement:** Track progress, measure success, and ensure accountability.
- **Stakeholder Communication:** Build investor confidence and promote transparency with detailed financial projections.
- **Strategic Planning:** Explore different strategic scenarios and support growth-related decisions.
- **Operational Efficiency:** Identify cost reduction opportunities and optimize business processes.
- **Regulatory Compliance:** Ensure accurate financial reporting and assess regulatory risks.

These exercises enable informed decision-making, efficient resource management, and strategic planning, helping organizations achieve their long-term objectives.

As financial and risk modeling is essential in any organization, architects frequently need to answer questions about **the (economic) value of technology investments** and architecture. Answering this question is a crucial skill for any senior architect, but it may take a lot of work to answer this seemingly harmless question concisely and convincingly to a non-technical audience.

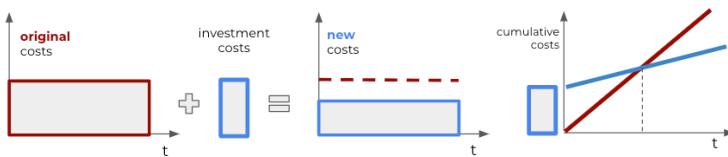
Having good architecture requires some investment. This investment is time and effort spent implementing some **architecture pattern**, reducing **technical debt**, or **refactoring code** to align with our architecture. Consequently, we need to explain the expected value of this investment.

In this post, I sketch two answers to the question of the economic value of architecture:

- the return-on-investment (ROI) metaphor
- the financial options metaphor

## 20.1: The Return-on-Investment Metaphor

In economic terms, **return on investment (ROI)** is a ratio between profits and costs over some period. In other words, ROI shows how much you **get back from your investment**. A high ROI means the investment's gains compare favorably to its cost. As a performance measure, you can use ROI to evaluate an investment's efficiency or compare the efficiencies of several different investments (Figure 1).



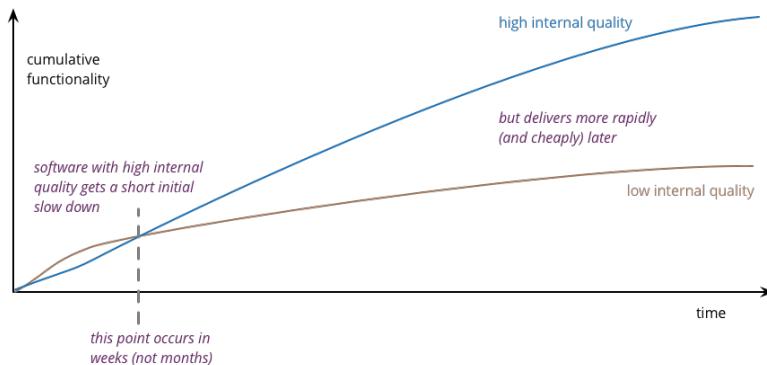
**Figure 1:** An illustration of the ROI metaphor. Investment leads to lower costs or higher value. It takes some time to reach a break-even point when an additional value has compensated for the investment. After the break-even point, we earn more than without the investment.

An investment in **good architecture** can help increase the ROI of IT. An excellent example of using the ROI metaphor to argue for investing in architecture is the post of Martin Fowler, who uses this argument to argue for the importance of **investing in improving internal quality**<sup>1</sup>. Figure 2 summarizes his argument.

Well-architect systems are typically much **easier to understand and change**. As our systems continuously evolve, the return on investing in making a system easier to understand and change can be significant. The primary value of such investment comes from generating **fewer errors and bugs**, more straightforward modifications, **short time-to-market**, and improved developer satisfaction.

---

<sup>1</sup><https://martinfowler.com/articles/is-quality-worth-cost.html>



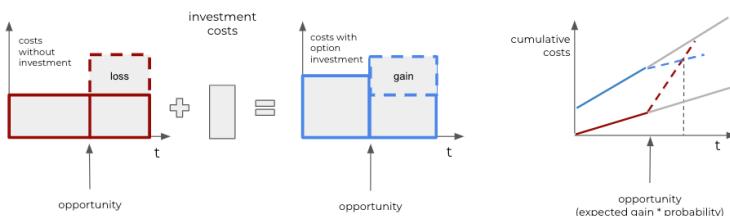
**Figure 2:** Software with high internal quality gets a short initial slowdown but delivers more rapidly and cheaply later (source [martinfowler.com/articles/is-quality-worth-cost.html](http://martinfowler.com/articles/is-quality-worth-cost.html)).

An ROI metaphor is easy to understand by a non-technical audience, but it has limitations to describe the value of architecture. The first limitation is that **measuring architecture, quality, and productivity is challenging**. Consequently, too much focus on ROI can lead to an obsession with cost-cutting. Costs are easy to measure, but the value of attributes like shorter time-to-market is much more difficult to quantify. Second, ROI is a good measure, but only some investments in architecture will increase profit. That is because we frequently have to make decisions with lots of uncertainty. Nevertheless, that does not mean that we should not make such investments. The following section explains why.

## 20.2: The Financial Options Metaphor

Gregor Hohpe has frequently argued that the best way to explain architecture to non-technical people is by using a **financial option metaphor**. A financial option is a **right, but not an obligation, to buy or sell financial instruments at a future point in time with some predefined price**. As such, a financial option is a **way to defer a decision**: instead of deciding to buy or sell a stock today, you have the right to make that decision in the future at a known price.

Options are **not free**, and a complex market for buying and selling financial options exists. Fischer Black and Myron Scholes managed to compute the value of an option with the [Black-Scholes Formula<sup>2</sup>](#). A critical parameter in establishing the option's value is the price at which you can purchase the stock in the future, the so-called **strike price**. The lower this strike price, the higher the value of the option (Figure 3).



**Figure 3:** An illustration of the financial option metaphor. Options have a price, leading to higher initial costs. However, if an opportunity can generate more value, we gain additional profit (or lose it if we do not invest).

Applying the financial option metaphor to IT architecture, we can argue that **buying options gives the business and IT a way to defer decisions**. Gregor Hohpe gives an example of the server size you need to purchase for a system. If your application is architected

<sup>2</sup>[https://en.wikipedia.org/wiki/Black%20-%20Scholes\\_model](https://en.wikipedia.org/wiki/Black%20-%20Scholes_model)

to be horizontally scalable, you can defer this decision: additional (virtual) servers can be ordered later at a known unit cost.

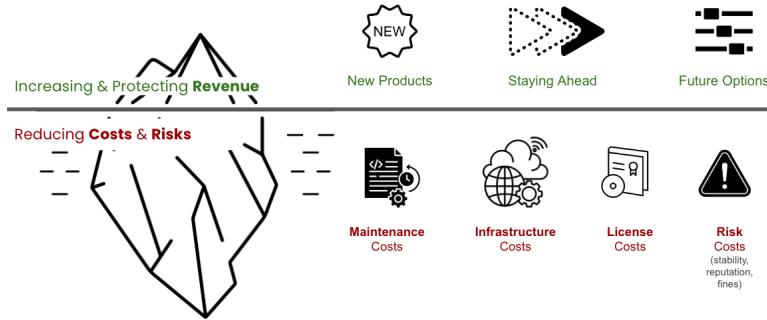
Another example of an IT option is architecting your system to **separate concerns**. For instance, deciding early what authentication mechanism an application should use may be challenging. A system that properly separates concerns allows changes to be localized so that updating one aspect of a system does not require expensive changing of the whole system. Such isolation will enable you to change a decision late in the project or even after go-live, at a nominal cost. For example, if authentication is a well-isolated concern, you will need to refactor only a minimal part of the system to use another authentication system.

The option's value originates from being able to **defer the decision until you have more information** while fixing the price. In times of uncertainty, the value of the options that architecture sells only increases.

As with any analogy, the financial options analogy has its limits. Again, it is **not easy to quantify** architecture values and have metrics for the value of separation of concerns or horizontal scaling. Second, while the metaphor may be easy to grasp for an economic audience, it may **require explaining** to other stakeholders, who may be less familiar with financial options markets.

## 20.3: A Communication Framework

In the end, I share a communication framework I developed and used to explain holistically the economic value of architecture and technology investment (Figure 4).



**Figure 4:** A framework for discussing investments and options.

I separate the value of investments in two buckets:

- Increasing and protecting revenue and
- Reducing costs and risks.

Increasing and protecting revenue investments have three forms:

- **Investments that create new revenue streams** by creating new products or adding new features. These investments are typically easier to defend and control, as most stakeholders intuitively understand that new functionality is needed to create new value for customers and generate more revenue. An essential aspect of this type of investment is tracking the product's success. Adding new features will not automatically create value for customers or revenue.
- **Investments needed to stay ahead.** This type of investment is a less obvious way to protect and increase revenue. It

boils down to the fact that you cannot stop developing your product as the rest of the world moves on. As the saying goes, “**It takes all the running you can do to keep in the same place.**” For instance, you must keep essential features in parity with the competition, your system must comply with changes in regulations, and your UX must be modern.

- **Investments needed to create future options** refer to being in shape to adapt to changes in the market more quickly and to bring new features to the market more quickly. Investing in keeping your system easy to maintain and extend directly creates more opportunities. Another way to look at this value driver is to frame it as preventing a revenue loss due to the impossibility of quickly adapting to future opportunities.

The second bucket relates to the more invisible part of the value created by investments:

- **Investments to reduce maintenance costs** need to ensure that your code is easy to understand, change, and test. Such investments directly reduce your most significant cost, people costs, as code that is easy to maintain requires fewer people (and other way around, see Figure 5). Alternatively, you can look at these investments as a way to spend more effort on innovation and creating new revenue streams rather than merely keeping the systems in the air.
- **Investments in reducing infrastructure costs** reduce spending and, if successful, are more directly visible. Such investments could take the form of redesigning your application to be more elastic, scaling up and down with minimal overhead. They could also create more transparency to have a precise image of all cost drivers and mechanisms to react quickly to any undesirable cost increases.
- **Investments in reducing license and vendor costs** ensure that there is no unnecessary diversity of technologies and vendor contracts and that you can leverage economies of

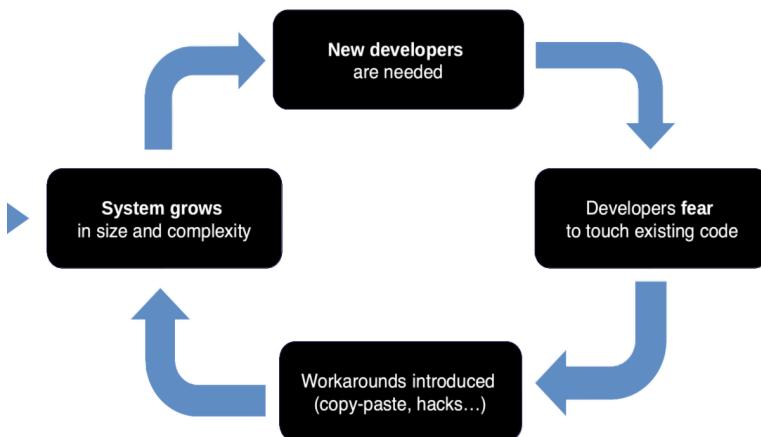
scale, as having fewer vendors with more users enables negotiating more favorable contracts.

- **Investments in reducing risk costs.** When your system is down, your business is disrupted, and you lose revenue. According to diverse studies<sup>3</sup>, the average cost of downtime ranges from \$2,300 to \$9,000 per minute. You must invest in keeping your system reliable and secure to avoid losing revenue and disrupting your business. While the benefits of these types of investments are huge, the challenge with building the business case for this investment is that a reliable system will only create a few incidents, making it less tangible for many stakeholders to understand the importance of continuing such investments. Or, as noted by Repenning and Sterman “[Nobody Ever Gets Credit for Fixing Problems that Never Happened](#)<sup>4</sup>”.

---

<sup>3</sup><https://www.atlassian.com/incident-management/kpis/cost-of-downtime>

<sup>4</sup>[https://web.mit.edu/nelsonr/www/CMR\\_Getting\\_Quality\\_v1.0.html](https://web.mit.edu/nelsonr/www/CMR_Getting_Quality_v1.0.html)



**Figure 5:** A downward spiral of poorly maintainable code. As such systems grow in size and complexity, more developers are needed to maintain them. If the system is not easy to maintain, people will avoid touching code as they can easily break it. This situation will lead to a workaround (such as copying and pasting code and diverse hacks). These inefficient workarounds further increase the size and complexity of code, requiring even more developers to maintain it. And the vicious cycle continues.

## 20.4: Questions to Consider

- *How can you effectively communicate the value of architectural investments to non-technical stakeholders in your organization?*
- *How do you weigh the importance of short-term cost reductions against long-term architecture improvements?*
- *How could the return-on-investment metaphor be useful in explaining the benefits of architecture investment to your team or organization?*
- *If you were to use the ROI metaphor to explain architecture's value to non-technical stakeholders, what examples or case studies would you use to illustrate your points?*
- *What are some potential pitfalls of relying too heavily on the ROI metaphor when deciding on architecture investments?*
- *How could you use the financial options metaphor to explain the value of architectural investments? What are the benefits and challenges of using this metaphor in your organization?*
- *How can you better quantify the value of architectural investments, particularly in terms of attributes like time-to-market and developer satisfaction?*
- *How might the financial options metaphor apply to recent decisions facing your organization or team, and how could it influence those decisions?*

## 21: Decision Intelligence in IT Architecture



image by istock

**IN THIS SECTION, YOU WILL:** Learn the basics of decision intelligence, the discipline of turning information into better actions, and its relevance for IT architecture practice.

**KEY POINTS:**

- Decision intelligence is the discipline of turning information into better actions.
- A decision involves more than just selecting from available options; it represents a commitment of resources you cannot take back.
- Many factors make the decision-making process more or less complex, such as the number of options, costs, cognitive load, emotions, and access to information.
- Data can significantly improve decision-making, but data do not guarantee objectivity and can even lead to more subjectivity.

**Decision intelligence** is a discipline concerned with selecting between options. It combines the best of **applied data science**, **social science**, and **managerial science** into a unified field that helps people use data to improve their lives, businesses, and the world around them. [Cassie Kozyrkov](#)<sup>1</sup> has popularized the field of decision intelligence and created several valuable resources to understand the decision-making process. I recommended her [posts](#)<sup>2</sup> and [online lessons](#)<sup>3</sup> to all architects because decision-making is an essential part of IT architects' job.

Now, in this and the next chapter, I want to share some golden nuggets I've gleaned from her teachings and how I've used them in the wild world of IT. IT architects, like decision-making ninjas, face critical choices every day. Here's how they flex their decision intelligence:

- By **making decisions** (e.g., deciding which cloud provider

---

<sup>1</sup>[https://en.wikipedia.org/wiki/Cassie\\_Kozyrkov](https://en.wikipedia.org/wiki/Cassie_Kozyrkov)

<sup>2</sup><https://www.linkedin.com/pulse/introduction-decision-intelligence-cassie-kozyrkov/>

<sup>3</sup><https://www.linkedin.com/learning/decision-intelligence/>

and services to use when moving applications from a private data center to a public cloud).

- By **creating mechanisms** for teams to make better decisions (e.g., [advisory forums<sup>4</sup>](#)).
- By **creating options<sup>5</sup>** for teams to make decisions later.

In every twist and turn, decision intelligence is the secret sauce that makes IT architects the unsung heroes of the tech world. So, grab your data-driven cape and dive into the art of smart decision-making!

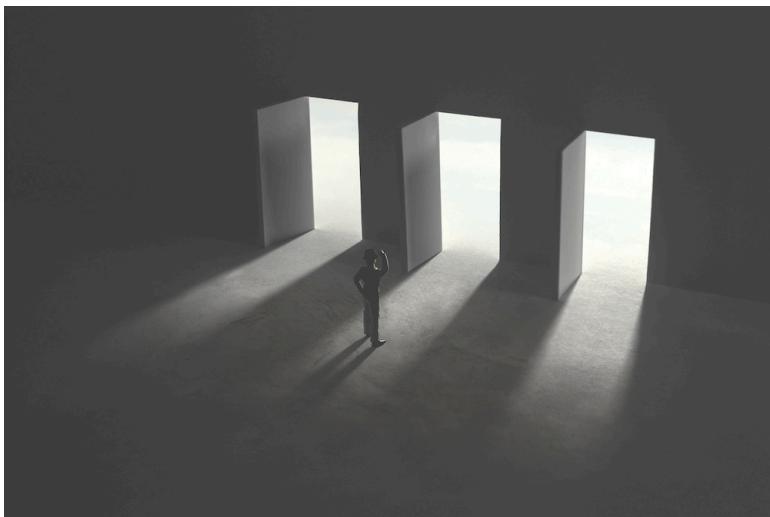


image by istock

---

<sup>4</sup><https://martinfowler.com/articles/scaling-architecture-conversationally.html>

<sup>5</sup><https://architectelevator.com/architecture/architecture-options/>

## 21.1: Basics of Decision-Making

Let's starts wit some basics: definition of decisions, outcomes, and goals.

### 21.1.1: Decision Is More Than Selecting Among Options

Kozyrkov defines a decision as more than just selecting from available options. A decision represents **an irrevocable allocation of resources**, which could be monetary, physical actions, time, or options. Whatever you decide to do, you will spend some time and other resources on it and will not get that time and resources back. The only way to reverse the consequences of some decisions is to invest more resources in that reversal.

Less obviously, optionality is also a resource. Choosing between two options may seem cost-free. However, the possibility of selecting some options is frequently lost once you decide. This loss of opportunity is considered an irrevocable allocation of resources. For instance, before starting a project in IT, you can select from many programming languages and frameworks to implement your system. However, after that, it is very costly to change that decision as you need to rewrite your system entirely in another language.

Having or losing options is directly related to a frequent topic of IT, a vendor lock-in, and it is one of the main drivers behind [creating or avoiding lock-in<sup>6</sup>](#). Lock-in in IT refers to a situation where a customer becomes dependent on a specific vendor for products and services, making switching to an alternative solution difficult, costly, or time-consuming. This dependency can result from factors such as proprietary technologies, high switching costs, contractual obligations, or the significant effort required to migrate data and systems.

---

<sup>6</sup><https://martinfowler.com/articles/oss-lockin.html>

From an IT architecture perspective, another important lesson of this view on decisions is that if there is no irreversible allocation of resources, we cannot talk about decisions. Ivory tower architects who make “principal decisions” that no one follows are technically not making any decisions.

### 21.1.2: Outcome = Decision x Luck

An outcome is a **result of a decision**. Two factors influence it:

- **the quality of the decision-making process** and
- **an element of randomness, or luck.**

We can only control our decision process. Luck? Well, that's like trying to control a cat—it's beyond our grasp and has its own agenda. Consequently, if we only consider the outcome, we can mistakenly attribute **good luck to good decision-making skills**, and bad luck to bad decision-making skills.

To fairly judge a decision, we need to look at the context and the information available when the decision was made. Imagine this: You're driving, and your GPS gives you two routes. One is 30 minutes shorter, so you take it. But 10 minutes in, a traffic jam from an accident makes you wish you had packed a lunch. You end up spending an extra hour stuck. Does this mean your decision was bad? No way! At the time, all signs pointed to a quick trip.

A recent prime example is the COVID-19 pandemic. The pandemic turned the global economy upside down, like a toddler with a snow globe. Some industries, like travel and tourism, took a nosedive (e.g., Uber, Booking.com, and [Airbnb](#)<sup>7</sup>). But on the flip side, COVID-19 gave a rocket boost to online tools and birthed a new era of virtual collaboration (think Zoom, Microsoft Teams, Slack, Miro).

---

<sup>7</sup><https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9998299/>

So remember, while we can't control the roll of the dice, we can master our decision-making process.

### 21.1.3: Economics of Decision-Making

I've often found myself tangled up in trivial decisions that sucked up all my time and energy. Not all decisions are worth that kind of investment. Enter the “[value of clairvoyance](#)<sup>8</sup>” concept (also known as the value of perfect information) in decision analysis. This nifty idea helps you figure out just how much effort, info, and resources you should throw at a decision.

For low-stakes decisions, perfectionism is like wearing a tuxedo to a beach party—completely unnecessary, and uncomfortable. On the flip side, high-stakes decisions deserve the royal treatment. According to the wise Cassie Kozyrkov, here's how to tackle decision-making like a pro:

1. **Visualize the Best and Worst Outcomes:** Start by picturing the potential paradise and disaster scenarios of your decision. This helps you grasp the stakes involved.
2. **Apply the “Value of Clairvoyance” Technique:** Imagine you've got a psychic on speed dial who can give you the perfect answer to your dilemma. How much would you pay for that crystal-clear insight? Think of the maximum amount of resources—money, time, or effort—you'd spend for this flawless foresight.
3. **Balance Investment with Importance:** This little exercise helps you figure out the true value of achieving perfect clarity and making the best possible choice.

If you realize that perfect information isn't worth much for a particular decision, it's time to trust your gut. This strategy helps

---

<sup>8</sup>[https://en.wikipedia.org/wiki/Value\\_of\\_information](https://en.wikipedia.org/wiki/Value_of_information)

you balance the effort you put into decision-making with the decision's actual importance.

For example, deciding on the best public cloud provider is like choosing a life partner—it's a high-impact decision that deserves thorough analysis. On the other hand, approving costs for an individual developer license that can be canceled at any time is like choosing what to have for lunch. Yet, companies often have procurement processes that make both these decisions feel like you're signing the Declaration of Independence.

So, next time you're stuck in a marathon meeting about whether to buy a €100 software library license, remember: not all decisions need to be treated like a royal decree. Save the deep dives for the big fish and keep the small fry simple!

## 21.2: Preparing for Making Decisions

Decisions are steering wheels for reaching our goals. Consequently, it is crucial to understand and define goals properly. But also to understand if there is a decision to be made at all.

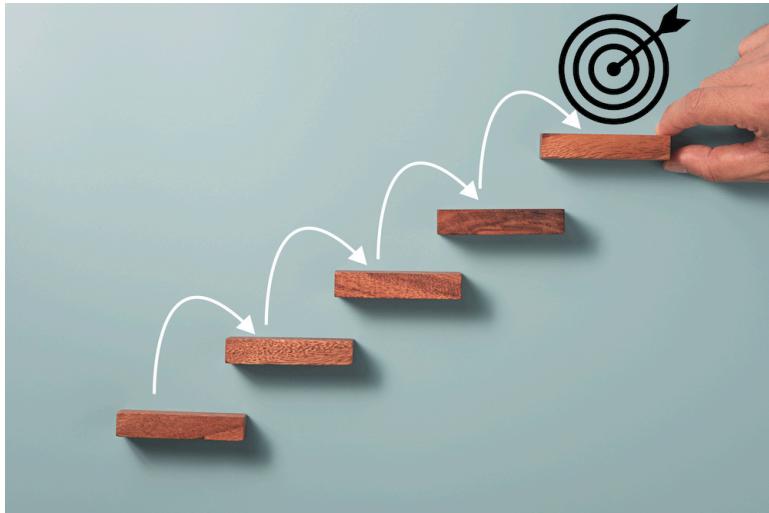


image by istock

### 21.2.1: Setting Goals

Practical goal setting is like trying to find your way out of an escape room—you need to understand your priorities and opportunities, or you'll end up running in circles. By identifying what's truly important and ignoring everyone else's distractions, you can focus on the stuff that really counts.

Common goal-setting blunders include making goals so vague that they float away like a helium balloon or so rigid that they shatter at the first sign of trouble. The secret sauce? **Layered goals** that bring clarity, each serving a different purpose. In the world of managerial

sciences, goals come in three flavors: outcome, performance, and process goals.

- **Outcome Goals:** These are the grand finales, the ultimate wins, but they can be as vague as a politician’s promise and influenced by things beyond your control. In business, it’s stuff like “creating value for customers” and “being profitable.” Nice, right? But how do you measure “value” and “profit” when you’re busy fighting off existential crises?
- **Performance Goals:** These goals are measurable and, if you’re not aiming for the moon, mostly within your control. In business, it’s all about increasing website visits, slashing infrastructure costs, and boosting revenue. They’re a bit aspirational and can be tricky to manage, but hey, no pain, no gain.
- **Process Goals:** They’re totally measurable and completely within your control. In business, this translates to rolling out new features on time or launching a targeted marketing campaign. These goals keep you on track, but they should always serve your higher aspirations.

You need all three types of goals, neatly connected like a well-oiled machine. For example, running a snazzy marketing campaign (a process goal) should attract more visitors to your site and boost revenue (performance goals), ultimately delivering more value to customers and fattening your bottom line (outcome goals). Consolidating IT infrastructure (a process goal) should trim overall costs (a performance goal), making your business more profitable (an outcome goal).

But beware! Don’t let process goals hog the spotlight and distract you from the big picture. Wise goal setting is all about layering your goals, aligning them with your priorities, setting ambitious targets, and establishing manageable processes, all while staying flexible and responsive to life’s curveballs.

## 21.2.2: Aligning Goals: The Principal-Agent Problem

One of the classic headaches in goal setting for complex organizations is the **principal-agent problem**. This nifty concept from economics is like a plot twist in a soap opera: the interests of the decision-maker (the agent) are as different from those of the owner (the principal). For example, the owners (principals) may be all about growth and expansion, while the managers (agents) might just be dreaming of longer lunch breaks and fatter paychecks. This clash of interests can lead to a mismanagement mess if not handled properly.

In the wild world of IT, a prime example is technology selection. Individual teams might want to use the latest, coolest tech based on their personal preferences. But letting each team run wild can turn your technology landscape into a tangled jungle. It's usually better for an IT organization to keep the tech menu limited. This way, it's easier to train newbies, maintain the codebase, and shuffle people between teams without causing a tech meltdown.

So, how do we get these misaligned interests on the same page? The principal needs to set up some **rules or constraints** to align the agent's decisions with their interests. This is kind of like giving your dog a fenced yard—freedom to play, but within safe boundaries.

This principle also applies to personal decision-making, especially when juggling short-term temptations and long-term goals. By setting up pre-emptive constraints, you can steer your choices toward those long-term dreams and avoid decisions that might look tempting now but are as regrettable as a midnight snack of expired sushi.

For instance, in the technology selection saga, one strategy I often

use is to create “golden paths<sup>9</sup>”—supporting a limited set of technologies and making it tougher to stray into uncharted territory. It’s like saying, “Sure, you can build with LEGO, but maybe let’s stick to this one box instead of the entire toy store.”

So remember, setting those golden paths and constraints isn’t about being a killjoy. It’s about keeping everyone aligned and avoiding a tech Tower of Babel.

### 21.2.3: Is There A Decision To Be Made?

In decision-making, especially when you’re not the one calling the shots, it’s crucial to figure out how to contribute effectively as the decision whisperer. First things first: determine if there’s even a decision to be made. Sometimes, the big cheese has already made up their mind and just needs you to rubber-stamp it like a bureaucratic formality.

Before you dive into the murky waters of faux decision-making, clarify whether there’s actually room for a decision. According to the Cassie Kozyrkov, this involves two simple steps.

1. **Check the Decision-Maker’s Auto-Pilot:** Figure out what the primary decision-maker would do if you weren’t in the picture. Would they carry on like a self-driving car?
2. **Deploy the Magic Question:** Ask them, “What would it take to change your mind?” If they reply, “Nothing!” then congratulations, you’ve just discovered you’re in a no-decision zone.

This latter question is your secret weapon for several reasons:

1. **Starts Insightful Conversations:** It’s like opening a treasure chest of insights into the decision-making process and the decision-maker’s mindset.

---

<sup>9</sup><https://engineering.spotify.com/2020/08/how-we-use-golden-paths-to-solve-fragmentation-in-our-software-ecosystem/>

2. **Identifies the Decision-Maker:** It helps you figure out if the person you're talking to is the real decision-maker or just a messenger.
3. **Detects Real Decisions:** If no information can change their mind, then there's no real decision to be made. You're just there to wave pom-poms and cheer for a pre-made choice.

In essence, this magical question helps you map out the decision-making landscape, gauge the decision-maker's openness to new ideas, and see if there's genuine room for making or influencing a decision. If their mind is more closed than a locked vault, you'll know it's time to save your energy for real decision-making battles.

So, next time you're in a meeting and wondering if you're there to make a difference or just to play the part of the wise advisor in a decision-making drama, remember to whip out the magic question. It's your ticket to knowing whether you're in for an epic decision-making saga or just a cameo appearance.



image by istock

## 21.3: Decision-Making Complexity

Some decisions are easier to make than others. Kozyrkov identifies 14 factors that can make decision-making more or less complex:

### 21.3.1: Number of options

Decision-making becomes simpler with **fewer options**. When choosing between a limited number of options, it's straightforward. However, as more options, especially combined choices, are introduced, the process becomes more complex, involving **compound decision-making** (several dependent decisions you must make together). The simplest scenario is a straightforward yes or no decision.

### 21.3.2: Clarity of options' boundaries

Some decisions are straightforward when the choice is **clear-cut**, like choosing your meal between a rock and an apple, where the preferable option is obvious. Deciding between two varieties of apples is slightly more challenging but remains easy if the decision **isn't significant or high-stakes**. In IT, having a preferred public cloud provider provides a clear-cut decision about public cloud hosting. Once inside a public cloud, selecting an appropriate service is much more challenging as hundreds of options are available.

### 21.3.3: Clarity of objectives

Having **clear objectives** is another factor that simplifies decision-making. It involves considering the most effective approach to the decision and quickly determining the criteria for making that choice.

### 21.3.4: Cost of making a decision

**Low-cost** of decision-making simplifies decision-making. These costs include the effort required to **evaluate** information, the ease of **implementing** the decision, and the potential **consequences of any mistakes**. Decisions are easier when these associated costs are low.

### 21.3.5: Costs of reversing a decision

While decisions typically involve a commitment of resources you can't undo, some decisions are considered reversible. If you can **change your decision quickly and with little cost**, the consequences of a wrong choice are less severe, making the decision easier.

### 21.3.6: Cognitive load

If a decision requires **significant mental effort**, such as remembering **many details** or making choices while **distracted**, it becomes more challenging. On the other hand, if you can make the decision easily and consistently, even amidst other tasks or distractions, then it's a simpler decision.

A lot of the work of IT architects involves creating visualizations and abstractions that can reduce cognitive load and make it easier to understand complex systems so that others can make better decisions about them.

### 21.3.7: Emotional impact

If a decision doesn't **evoke strong emotions** or significantly affect you emotionally, it tends to be easier. Conversely, decisions that stir up intense emotions or leave you highly agitated are more difficult.

For instance, the company's decision to use only one frontend programming language significantly affects people unfamiliar with the choice, as they cannot perform at a senior level in new technology for a long time and need to learn many new things. Many people negatively affected by this choice may decide to leave and find a job where they can work with technologies in which they are experts. So, a simple technological decision quickly becomes a personal career-making choice and an HR issue.

### **21.3.8: Pressure and stress**

Decisions made under conditions of **low pressure and stress** are generally easier, whereas those made in high-pressure, stressful situations are more challenging.

### **21.3.9: Access to information**

Decisions are easier when you **have complete and reliable information readily available**. In contrast, making decisions with only partial information and uncertain probabilities is more challenging. As discussed in the context of statistics, having limited information complicates the decision process as you need to guess missing pieces of information.

### **21.3.10: Risks and ambiguity**

Decisions become simpler when there is no **risk or ambiguity involved**. Risk and ambiguity, though different, both complicate decision-making. Ambiguity arises when the probabilities of outcomes are unknown, making choices uncertain. Risk, on the other hand, involves taking a known gamble, where you understand the potential consequences and likelihoods.

### 21.3.11: Timing

Difficulties arise when the decision's timing conflicts with other simultaneous decisions or when there's insufficient time to consider the choice thoroughly. Situations requiring a rapid response can add significant pressure, making the decision process more challenging.

### 21.3.12: Impact on others

Making decisions alone is generally easier. When you're the sole decision-maker, without involving others, and the decision's outcome impacts only you, the process is simpler. In contrast, making decisions in a social context is more complex. Factors like social scrutiny, considering the impact on others, balancing different preferences and opinions, and the potential effect on your reputation all add to the difficulty.

### 21.3.13: Internal conflicts

Decisions are more problematic when there are internal conflicts, as opposed to situations where all motivations and incentives align with the decision. For instance, deciding to make shortcuts in your systems design and skipping steps in a process to get some features quicker to the market vs. spending more time tidying and testing your code is a typical dilemma many software engineers and IT architects face.

### 21.3.14: Adversarial dynamics

Finally, adversarial dynamics impact decision-making. When you face competition or opposition from others, these decisions become more challenging compared to those made cooperatively or

independently. For example, when you merge two companies with different technology stacks (e.g., one using React and Java in AWS, and another Angular and C# in Azure) and want to consolidate on one stack, you may end up in competition and opposition with people from each company wanting a consolidated stack to be as close as possible to their previous one.

## 21.4: Decision-Making With Data and Tools

Decision-making has evolved beyond pen and paper, with data playing a crucial role in modern methods. Data, like the one I use in [Data Foundation](#), while visually appealing and powerful when used correctly, is **only a tool to assist in making informed decisions**. It's a means to an impactful end, but **without purposeful application, data is ineffective**.



image by istock

### 21.4.1: Remember that data has limitations

Just as not everything written in a book is true, data can be **misleading or incomplete**. It's a collection of information recorded by humans, subject to errors and omissions.

For instance, in the field of artificial intelligence (AI), **AI biases stem from the data it's fed**, reflecting the choices and prejudices of those who compile the data. The issues with AI bias are often due to poor decisions regarding data selection. **Data isn't inherently objective**; it carries the **implicit values of its creators**.

Data's value lies in its ability to **enhance memory, not ensure objectivity**. Embracing data means embracing a significant advancement in human potential. It's about transforming information into action, extending beyond the limits of personal memory to make better, more informed decisions.

### 21.4.2: Chose the right tool for the job

Cassie Kozyrkov breaks down the techniques for analyzing data into three snazzy groups<sup>10</sup>:

1. **Analytics:** This is like using data as a telescope that can give you a clear view of the available data landscape. It's like having a magic map that highlights viable options, reasonable assumptions, and thought-provoking questions. Data and Analytics can spark those "Aha!" moments by revealing insights that were previously hiding in plain sight. For those "What's the weather like?" kind of questions. Or "What is that service there in our public cloud costing as \$1M per year?"
2. **Statistics:** Think of this as your trusty Swiss Army knife for decision-making when you're dealing with the murky waters of incomplete information and uncertainty. For the "How likely am I to get struck by lightning?" type scenarios. Or "What is the probability of downtime or our IT services (famous three, four, five nines of availability)?"
3. **Machine Learning (ML)/AI:** This is your personal army of data minions, ready to tackle a gazillion decisions and mountains of data without breaking a sweat. For the "Can I build a weather-predicting robot?" level of inquiries. Or "What our IT costs will be next year be based on detailed traffic estimates for next year?"

---

<sup>10</sup><https://kozyrkov.medium.com/what-on-earth-is-data-science-eb1237d8cb37>

When you get the hang of these techniques, data transforms into your superpower, helping you ask sharper questions and deliver spot-on answers.

### **21.4.3: Prefer full information to partial information**

No matter which how you plan to use data, **full information is always preferable to partial information**. If you only have **partial information**, you're dealing with **uncertainty**, and that's where **statistical methods** come in.

Statistics is used when you don't have all the facts and need to manage uncertainty. They can help you balance the likelihood of a wrong decision against your data budget, considering your risk preferences.

### **21.4.4: Be in the driving seat**

Just staring at data and crunching numbers isn't decision-making. As the decision maker, your job is to set the stage, choose the relevant topics, frame the right questions, and guide the analysis like a maestro conducting an orchestra.

As a decision maker, it's crucial to **ask the right questions**, and figure out **which decisions are worth your brainpower**. Once you've identified those key decisions, then—and only then—should you whip out the advanced statistical or other methods to get more accurate answers in the murky waters of uncertainty. Remember, diving into data without asking the right questions is like wandering into a labyrinth with a blindfold on.

## 21.5: Questions to Consider

- *How do you typically approach decision-making in your professional role, and in what ways could you incorporate the principles of decision intelligence to enhance your decision-making process?*
- *Have you observed instances where excessive complexity in your organization resulted from poor decision-making? How can IT architects address this, and what role can they play in simplifying decision-making processes?*
- *Reflect on a recent significant decision you made. Were you aware of the resources you were committing and the opportunities you were preceding? How could you have evaluated these factors more effectively?*
- *Think of a situation where the outcome of a decision didn't align with your expectations. How did you judge the quality of the decision-making process in hindsight, and did you consider the role of luck or randomness?*
- *Consider a recent decision you faced. What would have been the value of perfect information in that scenario? How does this concept help you balance the effort and resources you allocate to different decisions?*
- *How do you set and align your goals, and what challenges have you faced in this process? Are there instances where misalignment has led to ineffective decision-making?*
- *What factors have you found to increase the complexity of decision-making in your experience? How do you manage these complexities effectively?*
- *Can you identify any habits or emotional factors contributing to your indecisiveness? What strategies can you employ to overcome these challenges?*
- *How do you use data in your decision-making process? Are there instances where data has misled your decisions, and how can you safeguard against this in the future?*

## **22: The Human Side of Decision-Making**



image by istock

**IN THIS SECTION, YOU WILL:** Learn the basic human factors influencing decision-making.

**KEY POINTS:**

- Decision-making is a human activity subject to human biases and limitations.
- Fundamental biases influencing decision-making include outcome, hindsight, and confirmation biases.
- Human intuition plays a vital role in decision-making.
- Group decision-making offers significant advantages but increases complexity as it requires higher decision-making skills from each member.

Cassie Kozyrkov<sup>1</sup>, in her [posts](#)<sup>2</sup> and [online lessons](#)<sup>3</sup> about design intelligence, often reminds us that decision-making is a uniquely human sport. It's a wild mix of brilliance, bias, and blunders. Just like how having a gourmet kitchen doesn't make you a master chef, having the best [data and IT tools](#) won't magically conjure up good decisions. IT architects need to remember that every decision is flavored by the quirks and quibbles of the human mind.

---

<sup>1</sup>[https://en.wikipedia.org/wiki/Cassie\\_Kozyrkov](https://en.wikipedia.org/wiki/Cassie_Kozyrkov)

<sup>2</sup><https://www.linkedin.com/pulse/introduction-decision-intelligence-cassie-kozyrkov/>

<sup>3</sup><https://www.linkedin.com/learning/decision-intelligence/>

## 22.1: Biases and Limitations

Fundamental biases influencing decision-making include outcome, hindsight, and confirmation biases.

### 22.1.1: Outcome Bias

The big trap in decision-making is falling into the **outcome bias** pit, where you **obsess over the results** rather than focusing on how you made the decision in the first place. If you trip over an unlucky result and think you picked the wrong option, you're learning how to fail with flair. This faulty thinking could make you make even wackier choices in the future.



image by istock

A decision and its outcome are like two different TV show episodes. The outcome is just what happens after the decision's cliffhanger. You can make a totally solid decision and still end up with a plot twist no one saw coming. Remember, outcomes are a cocktail mixed with decision-making, randomness, and a **splash of luck**. And luck, as we all know, is that elusive ingredient we can't control, often playing a starring role in our most complex dramas.

If we only focus on the outcome, neglecting the rich backstory and the information available at the time of the decision, we're akin to a film critic who only watches the ending. This can lead to **misjudging people's abilities**, rewarding or penalizing them based on a volatile mix of luck and skill. Therefore, it's crucial to discern whether someone's success stems from their astute decision-making or sheer chance. When evaluating decisions, it's essential not to be overly dazzled by the outcome.

In IT, outcome bias can come in many different forms, typically leading to the reinforcement of poor practices and processes based on the perceived successful outcome of projects:

#### **Example 1: Ignoring Best Practices Due to Success**

- A software team decides to release a critical update without performing adequate regression testing due to time constraints. The update is released and, fortunately, no major issues are found by the users.
- Because the update did not cause any immediate problems, the team and management might conclude that regression testing is not always necessary, reinforcing a risky behavior based on a lucky outcome rather than sound engineering practices.

#### **Example 2: Overlooking Security Flaws**

- A development team delivers a new web application feature without conducting a thorough security review. The feature gains popularity and does not encounter any immediate security incidents.
- The absence of immediate security breaches leads the team to believe that their approach to security is adequate, even though the feature might have significant vulnerabilities that have not yet been exploited. This can result in a continued lax attitude towards security best practices.

### **Example 3: Rewarding Speed Over Quality**

- A software project is completed and delivered significantly ahead of schedule, but with known technical debt and suboptimal code quality. The project is well-received by the client due to its timely delivery.
- Management praises the team for their speed, ignoring the long-term consequences of the technical debt. This could lead to future projects prioritizing speed over quality, increasing the risk of long-term maintenance issues and potential project failures.

### **Example 4: Hiring Decisions Based on Project Success**

- A software engineer is hired because they were part of a highly successful project at their previous company. The project's success was primarily due to market conditions and team effort rather than the individual's contributions.
- The hiring decision is heavily influenced by the success of the previous project, potentially overlooking the individual's actual skills and contributions. This could result in hiring someone who may not perform as expected in different circumstances, highlighting the risks of evaluating individual capabilities based on outcomes rather than detailed assessment.

### **Example 5: Skipping Code Reviews**

- A development team decides to skip code reviews to save time, and the software is delivered without any major bugs or issues.
- The team concludes that code reviews are not essential, leading to the institutionalization of skipping this important quality control step. Future projects might suffer from hidden bugs and poor code quality, which could have been caught during code reviews.

### Example 6: Misinterpreting Customer Feedback

- A feature is implemented based on limited customer feedback and is launched successfully. The positive reception leads the team to assume their approach to gathering and acting on feedback is effective.
- The team might conclude that extensive user research is unnecessary, believing that limited feedback is sufficient to guide development. This could result in future features missing critical insights from a broader user base, potentially leading to less successful outcomes.

It's important for teams to critically evaluate their methodologies and ensure that success is attributed to sound practices rather than favorable outcomes alone.

#### 22.1.2: Hindsight Bias

Looking back at decisions can be as tricky as explaining a magic trick once you know the secret, thanks to **hindsight bias**. Everything seems obvious in hindsight, even though it was as clear as mud at the time. The real way to judge a decision is to dig into the info and context available when it was made—like a detective piecing together clues.

Think about what the decision-maker considered, how they played detective gathering info, and where they got their facts. Also, check if they collected enough intel for the decision's importance or were winging it.

If you don't jot down this process, it's like trying to remember a dream—you'll miss many details and be left only with a vague feeling. This makes it tough to judge the quality of your decisions or anyone else's, stalling your growth as a decision-making wizard. Write down your decision-making process. This helps you determine if luck was messing with you or if your skills were on point.



image by istock

Hindsight bias in IT can lead to an oversimplified understanding of past events and decisions, creating an illusion of predictability. It is crucial to recognize the context and constraints under which decisions were made to learn accurately from past experiences and improve future practices:

#### **Example 1: Post-Mortem Analysis**

- A software project fails due to unexpected integration issues with third-party services.
- During the post-mortem analysis, team members and management claim that the integration issues were obvious and should have been anticipated. They overlook the fact that, at the time of decision-making, the integration appeared straightforward and the issues were not foreseeable with the information available.

#### **Example 2: Bug Discovery**

- A critical bug is discovered in the production environment that causes significant downtime.
- After the bug is found, developers and stakeholders assert that the bug was easy to spot and should have been caught during the testing phase. This perspective disregards the complexity and number of potential issues that testers were dealing with and that the bug was not apparent among the other potential problems at the time.

### **Example 3: Feature Failure**

- A new feature is released, but it fails to gain user adoption and is considered a failure.
- Team members and management might claim that they always had doubts about the feature's success and that it was destined to fail. This can lead to the erroneous belief that the failure was evident from the start, even if the decision to proceed with the feature was based on thorough research and positive initial feedback.

### **Example 4: Performance Issues**

- A software system experiences performance degradation under high load conditions that were not tested.
- After the performance issues arise, team members and stakeholders might say that it was clear the system would not handle high load and that stress testing should have been prioritized. This perspective ignores the fact that other pressing issues and constraints influenced the original decision-making process.

### **Example 5: Security Breach**

- A security vulnerability is exploited in a production system, leading to a data breach.

- Post-breach, it is often stated that the vulnerability was obvious and should have been addressed sooner. This belief neglects the context in which security measures were evaluated and the myriad of potential vulnerabilities that needed to be managed simultaneously.

#### **Example 6: Project Timeline Overruns**

- A software project exceeds its deadline due to unforeseen technical challenges.
- Once the challenges are known, team members might argue that the delays were predictable and that the original timeline was overly optimistic. This view ignores the uncertainty and the incomplete information available when the original timeline was set.

#### **Example 7: Technology Stack Decision**

- A particular technology stack is chosen for a project, but it later turns out to be ill-suited, causing significant rework.
- After the problems become apparent, developers and stakeholders might claim that it was clear from the beginning that this technology stack was not a good choice. They might forget that at the time of selection, the decision was made based on the best available information and the perceived advantages of the stack.

By acknowledging the role of hindsight bias, teams can foster a more realistic and fair evaluation of past projects and decisions.

#### **22.1.3: Confirmation Bias**

**Confirmation bias** is like having a pair of magical glasses that only let you see what you already believe. When you stumble upon a

new fact, your brain gives it a makeover to fit your beliefs, even before your morning coffee.

Being aware of this sneaky bias is essential because your brain loves to play tricks on you. It makes you think you're being objective while sneakily reinforcing your pre-existing ideas. This subconscious nudge can twist your understanding and decision-making without you even noticing.

Businesses are jumping on the data science bandwagon, hiring data scientists to make supposedly unbiased, data-driven decisions. But guess what? These decisions are often not as data-driven as they claim. For a decision to follow the data, it should be the data leading the dance, not your preconceived notions or biases—a simple idea harder to pull off than a flawless magic trick.

Confirmation bias is like your brain's way of playing a one-sided game of telephone with data. Even the most complex math won't save you if you still interpret results through your belief-tinted glasses. Extensive data analysis can end up being as useful as a screen door on a submarine if it's warped by confirmation bias. The real challenge is resisting the urge to twist the story after seeing the data. So, watch for your brain's tricks and let the data speak for itself—or risk your analysis being as misleading as a carnival funhouse mirror.



image by istock

Beating confirmation bias is like prepping for a magic show: you need a plan before the curtain goes up. Set clear objectives before you peek at the data. Consider what the data should mean to you beforehand so you don't get dazzled by surprising plot twists. This way, you can make genuinely data-driven decisions instead of falling into the trap of your brain's sneaky biases.

To really harness the power of data to generate questions and find answers, treat your dataset like a well-organized closet. Split it into two parts: one for analytics (exploration) and one for statistics (data-driven decisions), and never let them mingle. This separation is like keeping your superhero costume and secret identity separate—it stops you from shifting the goalposts once you've seen the data.

Confirmation bias in IT can lead to suboptimal decisions and hinder the effectiveness of problem-solving and innovation in IT:

#### Example 1: Tool Selection

- A development team prefers using a specific development

framework because of their past experiences with it.

- When choosing a framework for a new project, team members only seek out positive reviews and success stories about their preferred framework, ignoring or downplaying any negative feedback or alternative frameworks that might be better suited for the project's requirements.

### **Example 2: Debugging**

- A developer believes that a particular module is the source of a bug.
- The developer focuses exclusively on the suspected module, interpreting any evidence to support this belief, and may overlook or disregard indications that the bug originates from another part of the code. This can lead to extended debugging time and potentially missing the actual source of the issue.

### **Example 3: Code Reviews**

- A senior developer has a high opinion of a specific junior developer's skills.
- During code reviews, the senior developer may be more inclined to approve the junior developer's code with minimal scrutiny, interpreting any ambiguities or minor issues as acceptable or easily fixable, while being more critical of other developers' code.

### **Example 4: Performance Testing**

- A team is confident that their application will perform well under high load due to recent optimizations.
- When conducting performance tests, they may primarily focus on scenarios where they expect the application to perform well, ignoring or not thoroughly testing edge cases

or scenarios that might reveal performance bottlenecks. As a result, they might miss critical issues that only appear under certain conditions.

### **Example 5: Estimating Project Timelines**

- A project manager has a strong belief that the team can meet an aggressive deadline.
- The project manager seeks out and emphasizes information and past examples where similar deadlines were met, while ignoring or discounting instances where similar projects encountered delays. This can lead to unrealistic project timelines and potential burnout.

### **Example 6: User Feedback**

- The team has a preconceived notion that users will love a new feature they developed.
- When gathering user feedback, they may give more weight to positive comments and downplay or dismiss negative feedback. They might also ask leading questions that are likely to elicit positive responses, thus reinforcing their belief that the feature is well-received.

### **Example 7: Technology Adoption**

- A company decides to adopt a new technology stack based on industry trends and some initial positive experiences.
- The team focuses on success stories and favorable benchmarks that support the decision, while disregarding case studies or reports that highlight challenges and failures associated with the new technology. This can lead to underestimating the risks and difficulties of the adoption process.

It is important for teams to actively seek out and consider disconfirming evidence, adopt a critical thinking approach, and encourage diverse perspectives to mitigate the impact of confirmation bias. By being aware of this bias, teams can make more balanced and informed decisions, ultimately leading to better software outcomes.

#### 22.1.4: Other Human Limitations

In a classic behavioral economics study, researchers showed that **how you say something** can be more magical than a rabbit out of a hat. They gave decision-makers the same facts but with some wordplay—different wording. Despite the identical information, decisions did a Houdini act and varied wildly. A tweak in phrasing or tossing in unrelated details can make people's choices wobble like a tightrope walker in a windstorm. Our brains are suckers for cognitive biases and illusions, even when the cold, hard facts are staring us in the face.



image by istock

This means that dealing with data isn't as objective as we like to

think. How we mentally wrestle with data matters a lot. We may aim to use data to become more objective, but if we're not careful, it can pump up our pre-existing beliefs like a bouncy castle. Instead of uncovering new insights, we end up with our same old convictions, sabotaging our quest for objectivity and learning.

And let's be honest—your decision-making skills aren't precisely Olympic-level when you're **sleep-deprived, hungry, stressed**, or feeling the heat. These biological and emotional states can affect your ability to make top-notch decisions. Believing that sheer willpower or a PhD in decision-making will always lead to the best outcomes is like thinking you can win a marathon in flip-flops.

A jaw-dropping example is in the legal system: studies show that judges can hand out different sentences before and after lunch. Even these wise, experienced folks, whose decisions shape lives, can be swayed by something as simple as a rumbling stomach. This should be a big, flashing neon sign warning us about the limits and quirks of our decision-making processes. So next time you're about to make a big decision, grab a snack and a nap first!

## 22.2: Decisiveness

We frequently find ourselves stuck in the indecisiveness trap. Why? Well, people can be indecisive for all sorts of amusing reasons.

### 22.2.1: Bad Habits

First up, bad habits. A lot of folks don't realize that **dodging a decision is still a decision**. It's like standing in front of an ice cream shop, unable to choose a flavor until the shop closes, and boom—you've "decided" to go home ice cream-less. Delaying, postponing, or deprioritizing the decision-making process is just making an implicit choice.

For instance, if a company can't decide on using one consolidated tech stack for its systems, it ends up with a tech landscape messier than a toddler's playroom. That implicit decision comes with all the extra costs and complexities you'd expect.

### 22.2.2: Overwhelmed by Numerous Decisions

Then there's the classic "**too many choices**" dilemma. When faced with a smorgasbord of decisions, especially those of lower priority, our brains feel like an overstuffed suitcase. Our cognitive capacity is limited—we can't focus intensely on everything at once. If we spend too much mental energy deciding what color to paint the break room, we might leave no brainpower for the big decisions, like how to keep the servers from catching fire.

For instance, sometimes people treat IT architects like decision-making oracles, asking them to approve every trivial change (like upgrading a minor library version). This is like asking the CEO to decide the office snack rotation—it's a sure way to waste their strategic brainpower.

### 22.2.3: Emotions and Grief

Indecisiveness also loves to rear its head when emotions are involved, especially when all options are about as appealing as a soggy sandwich. When faced with **undesirable choices**, the practical move is to pick the least bad option. After thoroughly evaluating your choices and identifying the least awful one, it's time to bite the bullet and make the call.

People often get tangled in a web of grief or frustration when stuck with lousy options, hoping for a miracle solution that'll never come. It's like searching for unicorns in a petting zoo. Once it's clear that no better options will appear, it takes courage to move forward with the least terrible choice.

## 22.3: Intuition

Human intuition plays a vital role in decision-making. Robert Glass provided one of the best definitions of intuitions, describing it as a function of our mind that allows it to access a **rich fund of historically gleaned information** we are not necessarily aware we possess by a **method we do not understand** (Glass, 2006; page 125)<sup>4</sup>. Our unawareness of such knowledge does not mean we cannot use it.

In the context of decision-making, one of the main advantages of intuition is that accessing it is a **rapid process**, making **intuitive decisions straightforward**. Intuition is particularly useful for **low-value decisions** with low stakes, and a **quick resolution is preferable**. As we'll explore in future discussions on prioritization and decisiveness, seeking perfection in every decision is impractical due to limited time and energy. Therefore, it's essential to choose where to focus your efforts.

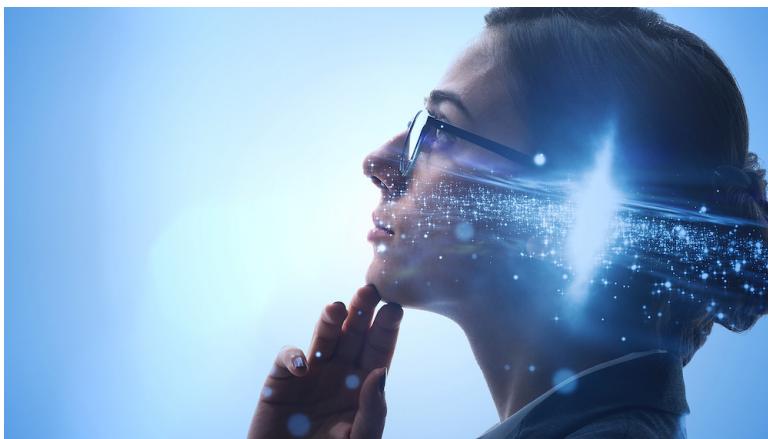


image by istock

Intuition is especially appropriate under certain conditions:

---

<sup>4</sup><https://www.amazon.com/Software-Creativity-2-0-Robert-Glass/dp/0977213315>

- **Time Pressure:** When time is limited and a detailed analysis isn't feasible, intuition can guide you.
- **Expertise:** If you have experience in a particular area, relying on intuition makes sense, as you've likely faced similar decisions before. In contrast, in unfamiliar contexts, intuition may not be reliable.
- **Unstructured Decisions:** Intuition can be valuable for decisions that lack a clear framework, like judging the quality of art. Expertise in the relevant field enhances the effectiveness of intuitive judgments.

Conversely, you should avoid relying on intuition too much in situations where more effort is warranted, including those with ample time, high importance, lack of expertise, and a structured decision-making process.

Intuition in IT can be a powerful tool when it is informed by experience and used in conjunction with data and thorough analysis. It can guide efficient problem-solving and quick decision-making in familiar contexts. However, overreliance on intuition without considering empirical evidence and diverse viewpoints can lead to significant mistakes, especially in unfamiliar or complex situations.

#### **22.3.0.1: Good Example: Intuitive Debugging**

**Scenario:** A seasoned software engineer is working on a complex system that suddenly starts behaving unexpectedly. The logs provide little information, and initial debugging efforts do not yield obvious clues.

##### **Good Use of Intuition**

1. **Pattern Recognition:** Drawing on years of experience, the engineer intuitively suspects that the issue might be related to a recent configuration change rather than a code issue.

2. **Focused Investigation:** The engineer quickly narrows down the potential causes based on this intuition, focusing on recent changes in the configuration files.
3. **Quick Resolution:** This intuition-driven approach leads to the discovery of a misconfiguration in a matter of minutes, saving the team hours of potentially fruitless debugging.

**Outcome:** The engineer's intuition, honed by experience, helps quickly identify and resolve the issue, demonstrating how intuition can efficiently guide problem-solving in complex scenarios and under time-pressure.

#### **22.3.0.2: Bad Example: Intuitive Decision on Technology Stack**

**Scenario:** A new project is starting, and the team needs to decide on the technology stack. The team lead has a strong intuitive preference for a specific new programming language and framework that they have used in the past.

##### **Bad Use of Intuition**

1. **Ignoring Data:** The team lead dismisses concerns and data presented by team members about the scalability and community support of the chosen technology. There was ample time to do proper analysis.
2. **Overconfidence:** Relying solely on personal intuition and past experience, the team lead pushes forward with the technology despite its known limitations for the project's specific needs.
3. **Future Problems:** As the project progresses, the team encounters significant issues related to performance and maintainability. These issues could have been mitigated or avoided by choosing a more appropriate technology stack based on objective criteria and thorough evaluation.

**Outcome:** The team lead's overreliance on intuition leads to a poor technology choice, resulting in increased technical debt, reduced productivity, and ultimately a less successful project. This example highlights how intuition, when used without adequate consideration of data and other perspectives, can lead to suboptimal decisions.

Balancing intuition with data-driven decision-making and collaborative input often leads to the best outcomes in software engineering.

## 22.4: Group Decision-Making Dynamics

Effective decision-making often involves recognizing that **you might not be the sole decision-maker**. In organizations, it's crucial to identify the actual decision-makers and understand how decision responsibility is distributed among them. Mastering this skill is essential for navigating organizational decision-making processes. It's important to question who really has the final say in decisions. In many cases, decision-making is more complex than it appears.



image by istock

**Group decision-making** offers significant advantages. While you might believe you have the best solutions, **incorporating diverse perspectives** can help **cover your blind spots**. Multiple decision-makers can **counterbalance the extreme tendencies** of an individual and compensate for **human limitations** like fatigue.

While group decision-making might sometimes constrain individual creativity, it also provides **safeguards against poor decisions** and **aligns individual motives with the organization's goals** (see the principal-agent problem in the next section). Having several independent decision-makers can align individual incentives with

the organization's needs, addressing this problem.

However, group decision-making isn't perfect. It **increases complexity** as it requires higher decision-making skills from each member. True **collaboration in decision-making** is more challenging than individual decision-making. It also tends to **slow down the decision process**.

Moreover, the benefits of group decision-making, like balancing individual biases, **rely on the independence of the decision-makers**. If everyone is in the same room, independence can be compromised by factors like **charisma or status**, potentially allowing the loudest voice to dominate, rather than the wisest.

Group settings can also **devolve into social exercises**, where **personal ego overshadows open-mindedness** to new information. Being aware of these pitfalls allows you to create rules that foster independent perspectives.

The **role of the note-taker** in group settings is also influential, as is the phenomenon of **responsibility diffusion**, where **unclear responsibilities** lead to reduced individual contribution.

In summary, **the more people involved in a decision, the higher the skill level required** to maximize the benefits and minimize the downsides of group decision-making. It's vital to structure the process to maintain independence, possibly by limiting decision-makers and increasing advisors. This approach distinguishes between making a decision and advocating for the execution of an already-made decision.

Group decision-making dynamics in IT can take various forms, including consensus, hierarchical, voting, and conflict resolution approaches. Group decision-making dynamics in IT can vary widely depending on the context, team structure, and decision at hand. Here are some examples illustrating different aspects of group decision-making dynamics in IT:

### **22.4.0.1: Example 1: Consensus Decision-Making for Technology Adoption**

**Scenario:** An IT team must decide which cloud platform to use for a new project. The options are AWS, Azure, and Google Cloud.

**Dynamics:**

1. **Information Sharing:** Team members share their experiences and knowledge about each platform. This includes presenting pros and cons, costs, and performance benchmarks.
2. **Brainstorming:** An open discussion is held where everyone is encouraged to voice their opinions and suggest potential solutions.
3. **Evaluation:** Each option is evaluated based on predefined criteria such as scalability, cost, ease of integration, and existing team expertise.
4. **Consensus Building:** The team works towards a consensus by discussing the trade-offs and attempting to agree on the platform that best meets the project's needs.
5. **Decision:** After a thorough discussion, the team decides to use AWS due to its robust ecosystem and familiarity with it.

**Influence:** Consensus decision-making ensures that all team members feel heard and can contribute to the decision, leading to higher buy-in and commitment to the chosen platform.

### **22.4.0.2: Example 2: Hierarchical Decision-Making for Security Policy**

**Scenario:** A decision must be made about implementing a new security policy to comply with regulatory requirements.

**Dynamics:**

1. **Top-Down Directive:** Senior management decides on the necessity of the new security policy based on compliance needs and risk assessments.
2. **Expert Input:** Security experts within the organization are consulted to provide detailed recommendations on the measures to implement.
3. **Implementation Plan:** The IT manager creates an implementation plan based on the expert recommendations and communicates it to the team.
4. **Team Execution:** The IT team is tasked with executing the plan, following the directives provided by management.

**Influence:** Hierarchical decision-making can be efficient, especially when quick, decisive action is required and the decision involves specialized knowledge. However, it may result in less buy-in from the team if they are not involved in the decision-making process.

#### **22.4.0.3: Example 3: Voting for Feature Prioritization**

**Scenario:** A software development team needs to prioritize features for the next release of their product.

**Dynamics:**

1. **Feature List:** A list of potential features is compiled based on customer feedback, market research, and internal brainstorming sessions.
2. **Discussion:** The team discusses the importance and impact of each feature, considering factors such as user value, development effort, and strategic alignment.
3. **Voting:** Each team member votes on their top features, often using a point system where they can allocate a certain number of points across the features.

4. **Ranking:** Features are ranked based on the total points received, and the top-ranked features are selected for the next release.

**Influence:** Voting democratizes the decision-making process and ensures that the prioritization reflects the team's collective opinion. This approach can enhance team morale and provide diverse perspectives are considered.

#### **22.4.0.4: Example 4: Conflict Resolution in Architecture Decisions**

**Scenario:** The development team is divided over whether to build a new application using a microservices architecture or a monolithic architecture.

##### **Dynamics**

1. **Initial Positions:** Team members present their initial positions, with some advocating for microservices due to their scalability and flexibility and others for a monolithic architecture due to its simplicity and ease of deployment.
2. **Evidence Gathering:** Both sides present evidence, including case studies, technical articles, and expert opinions, to support their arguments.
3. **Facilitated Discussion:** A neutral facilitator (such as an architect) leads a structured discussion to explore the pros and cons of each approach.
4. **Compromise and Integration:** The team seeks a compromise or an integrated solution, such as starting with a monolithic architecture and planning to evolve to microservices as the application grows.
5. **Final Decision:** After thorough discussion and consideration of all viewpoints, the team decides to balance immediate needs with future scalability.

**Influence:** Structured conflict resolution ensures that all voices are heard and helps the team make a well-considered decision. By combining the strengths of different viewpoints, this process can enhance mutual understanding and lead to better decisions.

Each method has its advantages and can be suitable for different decisions. Understanding these group dynamics can help teams navigate complex decisions more effectively, leading to better outcomes and stronger team cohesion.

## 22.5: Questions to Consider

- *How do your personal biases, such as outcome, hindsight, and confirmation bias, influence your decision-making process? Can you identify a recent decision where these biases might have played a role?*
- *Reflect on a situation where the outcome was bad, but the quality of your decision-making process was solid. How did you respond to this outcome, and what lessons did you learn?*
- *In what ways do you think hindsight bias has affected your ability to evaluate past decisions accurately? Can you think of a decision that seemed obvious in retrospect but was unclear at the time?*
- *Consider a decision you made recently. Did you document the decision-making process? If not, how could documenting this process help you in evaluating your decisions more effectively in the future?*
- *How does confirmation bias impact your interpretation of new information? Can you recall an instance where you ignored or misinterpreted data to fit your pre-existing beliefs?*
- *Think about a decision where changing your perspective or how information was presented (e.g., through different wording) might have led you to a different conclusion. How does this realization affect your approach to decision-making?*
- *Reflect on a time when your physical or emotional state might have influenced a decision. What does this tell you about the importance of being aware of your condition when making decisions?*
- *Consider a decision where intuition played a significant role. Was the decision effective, and would you rely on intuition under similar circumstances in the future?*
- *How do you balance the benefits of group decision-making with its challenges, such as social dynamics and the diffusion of responsibility?*

# **Part IV: Wrapping Up**

## **23: Summary**



image by gerd altmann from pixabay

**IN THIS SECTION, YOU WILL:** Look back at the content in this book.

**KEY POINTS:**

- This playbook aims to share my approach to running an IT architecture practice in larger organizations based on my experience as Chief Architect at AVIV Group, eBay Classifieds, and Adevinta.
- I called this approach “Grounded Architecture,” highlighting the need for any IT architecture function to stay well-connected to all levels of an organization and led by data.
- In this section, I summarize the key points from the playbook.

This book shared my approach to running an IT architecture practice in larger organizations based on my experience at AVIV Group, eBay Classifieds, and Adevinta. I called this approach “Grounded Architecture”—architecture with strong foundations and deep roots. Prioritizing people interactions and data over processes and tools, Grounded Architecture aims to connect architecture practice to all organizational parts and levels, serving as an antidote to the “ivory tower” architecture.

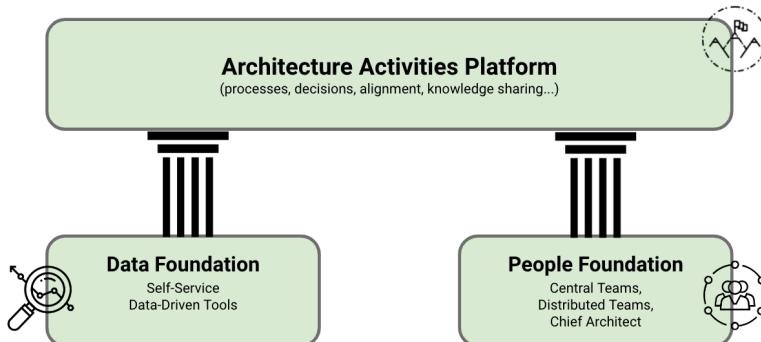
This book introduced Grounded Architecture as an IT architecture practice with two main parts:

- **Structure**, elements you need to have to run Grounded Architecture practice, and
- **Guiding Principles**, pieces of advice that can help put the ideas of Grounded Architecture into practice.

Figure 1 shows the structure of the Grounded Architecture consisting of three elements:

- **Data Foundation**,

- People Foundation,
- Architecture Activities Platform.



*Figure 1: The structure of Grounded Architecture.*

The *Data Foundation* ensures that architects can make data-informed decisions based on a real-time and complete overview of the organization's technology landscape.

The *People Foundation* is another essential element of Grounded Architecture. A strong network of people doing architecture across the organization is crucial to ensure that architecture function has any tangible impact.

Lastly, the *Architecture Activities Platform* defines a set of processes and agreements enabling architects to do everything architecture typically does, leveraging data and People Foundations to create a data-informed, organization-wide impact.

As a part of my work on Grounded Architecture, I also provided several guiding principles and tools that I found helpful to introduce the ideas of Grounded Architecture in practice. I grouped these resources into two parts:

- Being an Architect:
  - Architects as Superglue

- Skills
- Impact
- Leadership
- Architects' Career Paths

- Doing Architecture: Inspirations:

- Culture Map: Architects' Culture Mindfield Compass
- Managing Organization Complexity: Six Simple Rules
- Product Development and The Build Trap
- Architecture Governance: Mandates, Taxation, Nudge
- Economic Modeling: ROI and Financial Options
- Decision Intelligence in IT Architecture
- The Human Side of Decision-Making

When Grounded Architecture is in place, it can have a significant positive impact on the functioning of an organization:

- Enable Execution of Architecture Function At Scale,
- Increase the Quality of Decision-Making with Data,
- Maximize Organizational Alignment,
- Maximize Organizational Learning, and
- Increase Architecture Function Adaptivity.

While you may borrow ideas from others, every organization is different, and your approach must adapt to your context. When forming architecture functions, I always advise not to forget these two pieces of advice from Gregor Hohpe<sup>1</sup>:

- “Your architecture team’s job is to solve your biggest problems. The best setup is the one that allows it to accomplish that.”
- “Your organization has to earn its way to an effective architecture function. You can’t just plug some architects into the current mess and expect it to solve all your problems.”

---

<sup>1</sup><https://architectelevator.com/architecture/organizing-architecture/>

## 24: Cheat Sheet

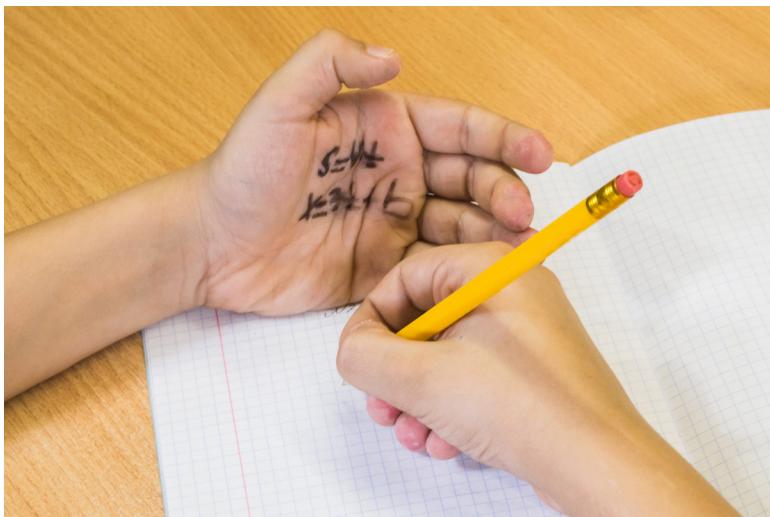


image by istock

**IN THIS SECTION, YOU WILL:** Get a cheatsheet with all key points from in all sections.

## 24.1: Introductions

### 24.1.1: Introduction

- This book will share my approach to running an IT architecture practice in larger organizations based on my experience at AVIV Group, eBay Classifieds, and Adevinta. I call this approach “Grounded Architecture”—architecture with strong foundations and deep roots.
- Prioritizing people interactions and data over processes and tools, Grounded Architecture aims to connect architecture practice to all organizational levels as an antidote to the “ivory tower” architecture.
- I also explain my motivation to write this book.

### 24.1.2: Context: Fast-Moving Global Organizations

- To better understand any idea or solution, it is crucial to understand the context in which this idea developed.
- The Grounded Architecture approach has evolved in the context of global, loosely coupled organizations that are diverse, with nonlinear growth dynamics, and under transformation pressures.

### 24.1.3: Goals

- I identified the following needs that an architecture function should support: Executing At Scale, Adaptivity, Improving the Quality of Decision-Making with Data, and Maximizing Organizational Alignment & Learning.

## 24.2: Grounded Architecture: Introduction

### 24.2.1: Data Foundation

- The architecture Data Foundation serves as a medium to create a complete, up-to-date picture of critical elements of the technology landscapes of big organizations.
- The Data Foundation provides an architecture-centric view of data about a technology landscape based on source code analyses, public cloud billing reports, vibrancy reports, or incident tickets.
- To facilitate the creation of a Data Foundation, I have been working on creating open-source tools that can help obtain valuable architectural insights from data sources, such as source code repositories.

### 24.2.2: People Foundation

- Developing the architecture function requires having competent, empowered, and motivated architects. Architecture practice must carefully organize, empower, and leverage scarce talent.
- In my work in the past few years, I combined two teams of architects: a small central architecture team and a cross-organizational distributed virtual team.

### 24.2.3: Architecture Activities Platform

- The Architecture Activities Platform is a system of processes and agreements enabling architects to do everything architec-

ture typically does, leveraging Data and People Foundations to create a data-informed, organization-wide impact.

- Examples of activities include: supporting teams in their daily work; tracking tech debt, defining tech debt reduction programs; performing technical due diligence; standardizing processes and documentation; defining cloud, data, and platform strategies.

#### **24.2.4: Value of Grounded Architecture Structure**

- When a Grounded Architecture structure is in place, it can significantly impact an organization's functioning.
- These impact categories are Executing At Scale, Improving the Quality of Decision-Making with Data, Maximizing Organizational Alignment & Learning, and Higher Adaptivity.

## 24.3: Being Architect: Introduction

### 24.3.1: Architects as Superglue

- Architects in IT organizations should develop as “superglue,” people who hold architecture, technical details, business needs, and people together across a large organization or complex projects.
- Architects need to be technically strong. But their unique strengths should stem from being able to relate technical issues with business and broader issues.
- Architects should stand on three legs: skills, impact, and leadership.

### 24.3.2: Skills

- A typical skillset of an architect includes hard (technical) skills, soft (people & social) skills, product development, business skills, and decision-making skills.

### 24.3.3: Impact

- Architects' work is evaluated based on their impact on the organization.
- Architects can make an impact via three pillars: Big-Picture Thinking, Execution, and Leveling-Up.

### **24.3.4: Leadership**

- My view of architecture leadership is inspired by David Marquet's work and Netflix's valued behaviors.
- Marquet focused on leadership and organizational management, particularly emphasizing the principles of Intent-Based Leadership.
- Borrowing from Netflix's original values, the following behavioral traits are crucial for architects: communication, judgment, impact, inclusion, selflessness, courage, integrity, curiosity, innovation, and passion.

### **24.3.5: Architects' Career Paths: Raising the Bar**

- Architects' career paths ideally stem from a strong engineering background.
- Hiring architects requires constantly raising the bar to ensure a strong and diverse team structure.

## 24.4: Doing Architecture: Introduction

### 24.4.1: Decision Intelligence in IT Architecture

- Decision intelligence is the discipline of turning information into better actions.
- A decision involves more than just selecting from available options; it represents a commitment of resources you cannot take back.
- Many factors make the decision-making process more or less complex, such as the number of options, costs, cognitive load, emotions, and access to information.
- Decision-making is a human activity subject to human biases and limitations. Fundamental biases influencing decision-making include outcome, hindsight, and confirmation bias.
- Data can significantly influence decision-making. But data do not guarantee objectivity and can sometimes lead to even more subjectivity.

### 24.4.2: The Culture Map: Architects' Culture Mindfield Compass

- I have found the work of Erin Meyer, The Culture Map, to be a beneficial tool for architects to work harmoniously with people from various cultures and backgrounds.
- Meyer's model contains eight scales, each representing a key area, showing how cultures vary from extreme to extreme: Communicating, Evaluating, Persuading, Leading, Deciding, Trusting, Disagreeing, and Scheduling.

### **24.4.3: Managing Organizational Complexity: Six Simple Rules**

- The Six Simple Rules approach emphasizes that in today's complicated business environment, you must set up organizational structures based on cooperation.
- To deal with complexity, organizations should depend on the judgment of their people and on these people cooperating.
- This view is well aligned with the ideas of Grounded Architecture.

### **24.4.4: Understanding Product Development**

- When it comes to product development, I generally recommend two resources for architects: "Escaping the Build Trap: How Effective Product Management Creates Real Value" by Melissa Perri and "The Discipline of Market Leader" by Michael Treacy and Fred Wiersema.
- The build trap occurs when businesses focus too much on their product's features and functionalities, overlooking customers' needs and preferences.
- The Discipline of Market Leader highlights three strategic paths a company can use to achieve market leadership: operational excellence, product leadership, and customer intimacy.

### **24.4.5: Architecture Governance: Nudge, Taxation, Mandates**

- Grounded Architecture supports governance models adaptable to organizations' complex and diverse needs. A technology governance model should be a well-balanced hybrid of three different styles of governing: mandates and bans, taxes, and nudging.

- Nudging is a form of governing where you create subtle or indirect suggestions influencing someone's behavior or decision-making without forcing them or limiting their freedom of choice.
- Governing with taxes (economic incentives) is a form of guiding in which people are not forbidden to make some decisions but need to "pay" some form of taxes on used resources.
- With mandates and bans, you guide people by explicitly defining what they should or should not do.

#### **24.4.6: Economic Modeling: ROI and Financial Options**

- Architects are frequently asked about the (economic) value of architecture or technology investments.
- Answering this question is a crucial skill for any senior architect. But it may be difficult to answer this seemingly harmless question concisely and convincingly to a non-technical audience.
- Borrowing from existing literature, I sketch two answers to the question of the economic value of architecture: the return on investment metaphor and the selling options metaphor.

# **Part V: To Probe Further**

## 25: Bookshelf

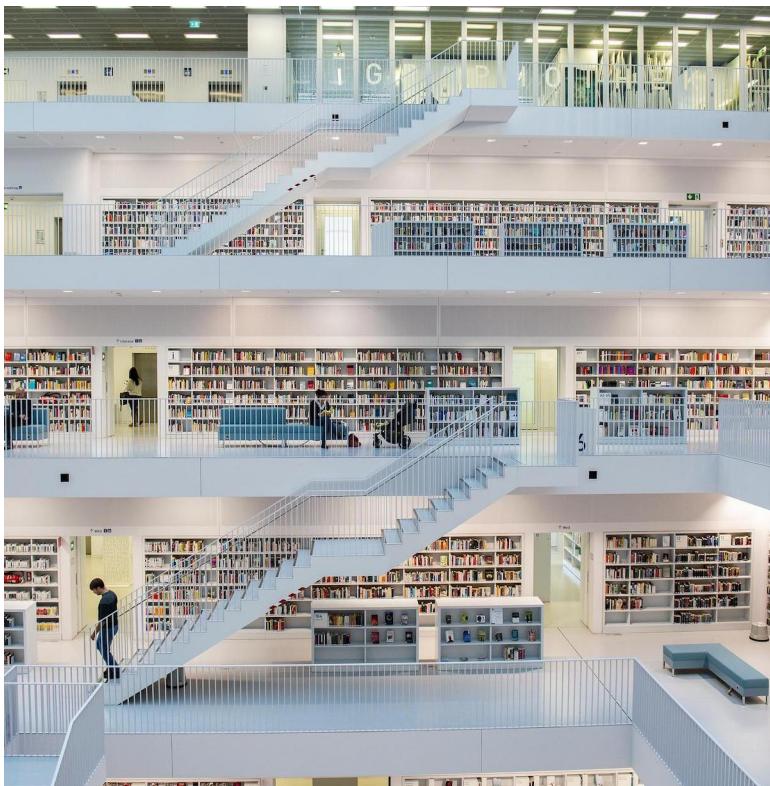


image by oli gotting from pixabay

**IN THIS SECTION, YOU WILL:** Get an overview of the background work to probe further, linking several external resources inspiring my work.

## 25.1: Introduction



**The Software Architect Elevator: Redefining the Architect's Role** by Gregor Hohpe defines architects as people that can fill a void in large enterprises: they work and communicate closely with technical staff on projects but are also able to convey technical topics to upper management without losing the essence of the message. Conversely, they understand the company's business strategy and can translate it into technical decisions that support it.

---



**SE Radio: On Architecture**<sup>1</sup>: for those interested in IT Architecture, I've created a curated Spotify playlist of Software Engineering Radio Episodes focusing on Architecture.

---

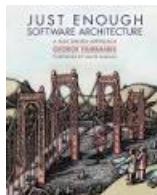
<sup>1</sup><https://open.spotify.com/playlist/7GVD86edcILnVPjsZFTy28?si=f44d0bef360e4818>



**Software Architecture for Developers** by Simon Brown is a practical, pragmatic, and lightweight software architecture guide specifically aimed at software developers.

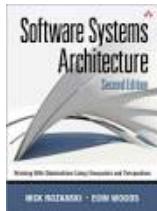


**Software Architecture: The Hard Parts** is structured as a narrative about a team breaking down a faulty, outdated monolithic application into a modern microservices-based architecture. The authors compare different aspects of how a monolithic architecture might have been written to do something in the past, then how modern microservice architecture could do the same thing today, offering advice and approaches for practical trade-off analysis when refactoring a large monolith app.



**Just Enough Software Architecture** by George Fairbanks is an approachable and comprehensive book.

---



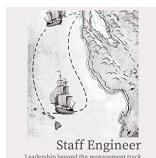
**Software Systems Architecture** by Nick and Eóin is another practitioner-oriented guide to designing and implementing effective architectures for information systems.

## 25.2: Career Development



**The Staff Engineer's Path** by Tanya Reilly. Similar to my view of architects, the staff engineer's path allows engineers to contribute at a high level as role models, driving big projects, determining technical strategy, and raising everyone's skills.

---

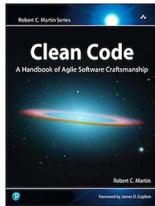


**Staff Engineer: technical leadership beyond the management track** by Will Larson defined technical leadership beyond the management track. I share many of the views presented in this book regarding the development and skills of architects.

## 25.3: Hard Skills

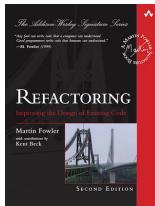
This section is heavily influenced by Gregor Hohpe's article [The Architect's Path - Part 2 - Bookshelf](#)<sup>2</sup>

---



**Clean Code** by Bob Martin discusses good code, and good code is clean. The basics of naming, functions that do one thing well, and formatting.

---



**Refactoring: Improving the Design of Existing Code** by Martin Fowler. Good software evolves, gains entropy, and is then restructured. Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior.

---

<sup>2</sup><https://architectelevator.com/architecture/architect-bookshelf/>



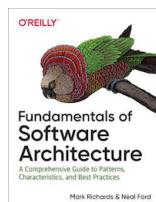
**Design Patterns** by Gamma, Helm, Johnson, and Vlissides. Design patterns help us make balanced decisions on the design of our code.

---

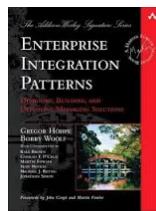


**Building Microservices (2nd Edition)** by Sam Newman is a book about architectural trade-offs and considerations in distributed system design.

---



**Fundamentals of Software Architecture** by Neal and Mark covers soft skills, modularity, component-based thinking, and architectural styles.



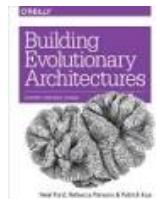
**Enterprise Integration Patterns** for anyone trying to connect systems without coupling them too tightly.

---

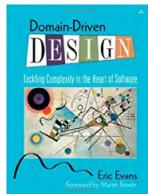


**Pattern-Oriented Software Architecture:** one of the most comprehensive references for distributed system design.

---



**Building Evolutionary Architectures** describes how fitness functions can guide architectural change over time.



**Domain-Driven Design** by Eric Evans promotes the idea that to develop software for a complex domain, we need to build Ubiquitous Language that embeds domain terminology into the software systems that we build. DDD stresses defining models in software and evolving them during the software product's life.



**Release It! (2nd Edition)** by Mike Nygard's is about architecture stability decisions and how to build “cynical” software. Besides the well-known stability patterns like circuit breakers and bulkheads, the book introduces foundational knowledge about running systems in production, from processes to network to security, also covering more process-oriented questions like deployments or handling security.



**Designing Data Intensive Systems** by Martin Kleppmann is a mini-encyclopedia of modern data engineering. The book lays down the principles of current distributed big data systems. It covers replication, partitioning, linearizability, locking, write skew, phantoms, transactions, event logs, and more.

---



**Thinking in Systems** by Donella Meadows describes a system as more than the sum of its parts. It may exhibit adaptive, dynamic, goal-seeking, self-preserving, and sometimes evolutionary behavior.



**Don't Make Me Think, Revisited:** a common sense approach to Web usability by Steve Krug provides a practical guide for understanding web usability and user experience.

---

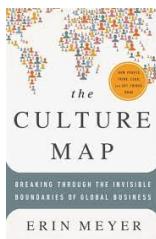


**System Design Interview (Volume 1 & 2)** by Alex Xu. A “real-world” systems design book is not just for preparing for the systems design interview but also for strengthening your systems design muscle for the day-to-day architectural work.

## 25.4: Soft Skills



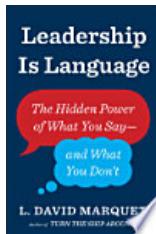
**Team Topologies** by Manuel Pais and Matthew Skelton describes four fundamental topologies (stream-aligned teams, enabling teams, complicated subsystem teams, and platform teams), and three fundamental interaction modes (collaboration, X-as-a-Service, and facilitation).



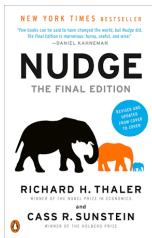
**The Culture Map: Decoding How People Think, Lead, and Get Things Done Across Cultures:** provides a framework for handling intercultural differences in business and illustrates how different cultures perceive the world. Awareness of intercultural differences is crucial for the success of an architect in an international setting. It helps us understand these differences and, in doing so, improves our ability to react to specific behaviors that might have once seemed strange.



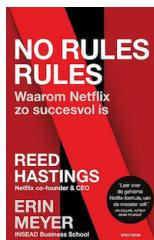
**Turn the Ship Around!** by L. David Marquet: Around reveals how one United States Navy captain managed to turn a dissatisfied submarine crew into a formidable and respected team. By changing how we think about leadership, this story will show you that we all have the power to be leaders inside.



**Leadership is Language** by L. David Marquet is a playbook for successful team management. It teaches leaders to change their language and mindsets to improve decision-making, empower workers, and achieve better results.



**Nudge** by Richard Thaler draws on psychology and behavioral economics research to define libertarian paternalism as an active engineering of choice architecture. The book also popularised the concept of a nudge, a choice architecture that predictably alters people's behavior without restricting options or significantly changing their economic incentives.

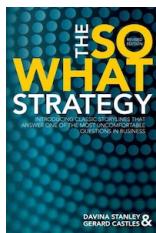


**No Rules Rules** by Erin Meyer and Reed Hastings provides insights into Netflix culture. When you give low-level employees access to information generally reserved for high-level executives, they get more done independently. They work faster without stopping to ask for information and approval. They make better decisions without needing input from the top.



**Dare To Lead** by Brené Brown defines a leader as anyone who takes responsibility for finding the potential in people and processes and dares to develop that potential.

---

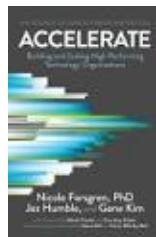


**The So What Strategy** by Davina Stanley and Gerard Castles. Any architect has been caught at the end of a presentation when their audience, perhaps a leadership team or a Steering Committee, looks at them blankly and asks this most uncomfortable question: 'So what?' How does that help? The book provides a pragmatic approach to answering that question in one powerful sentence. Or how to set yourself up so nobody asks it.



**Never Split the Difference** by Chris Voss. Negotiations occur in many fields, such as business, and in some critical situations, like hostage situations. The book is a guide on how to behave best when certain things happen, regardless of whether that includes the need for negotiation techniques in hostage situations or in business (and architecture).

## 25.5: Organization and Processes

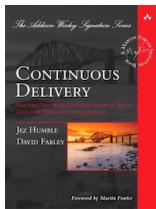


**Accelerate** — acceleration is the second derivative of position (speed being the first), so if you want to move faster, you need to accelerate. Sometimes, you need to jerk the system a bit, which is the proper term for the third derivative.

---

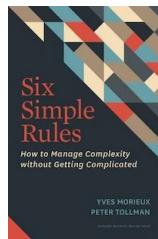


**The Phoenix Project** and **The Unicorn Project** by Gene Kim  
The Phoenix Project is a best-selling novel about DevOps. The book's characters reveal through their actions why it's so important for organizations to put security first and tear down the silos that have traditionally existed between development and operations teams.



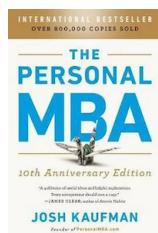
**Continuous Delivery** – through automation of the build, deployment, and testing process, and improved collaboration between developers, testers, and operations, delivery teams can get changes released in a matter of hours-sometimes even minutes-no matter what the size of a project or the complexity of its code base.

## 25.6: Business, Product, Strategy



**Six Simple Rules: How to Manage Complexity without Getting Complicated** by Yves Morieux and Peter Tollman: the book emphasizes that in today's complicated business environment, you must set up organizational structures based on cooperation. To deal with complexity, organizations should depend on the judgment of their people, which requires giving them more autonomy to act. It also depends on these people cooperating to utilize the organization's capabilities to cope with complex problems.

---



**The Personal MBA 10th Anniversary Edition** by Josh Kaufman provides an overview of the essentials of every major business topic: entrepreneurship, product development, marketing, sales, negotiation, accounting, finance, productivity, communication, psychology, leadership, systems design, analysis, and operations management.



**Technology Strategy Patterns** by Eben Hewitt provides a shared language—in the form of repeatable, practical patterns and templates—to produce great technology strategies.

---



**Escaping the Build Trap** by Melissa Perri is a practical guide to product management, product strategy, and ensuring product success.

## 26: Tools

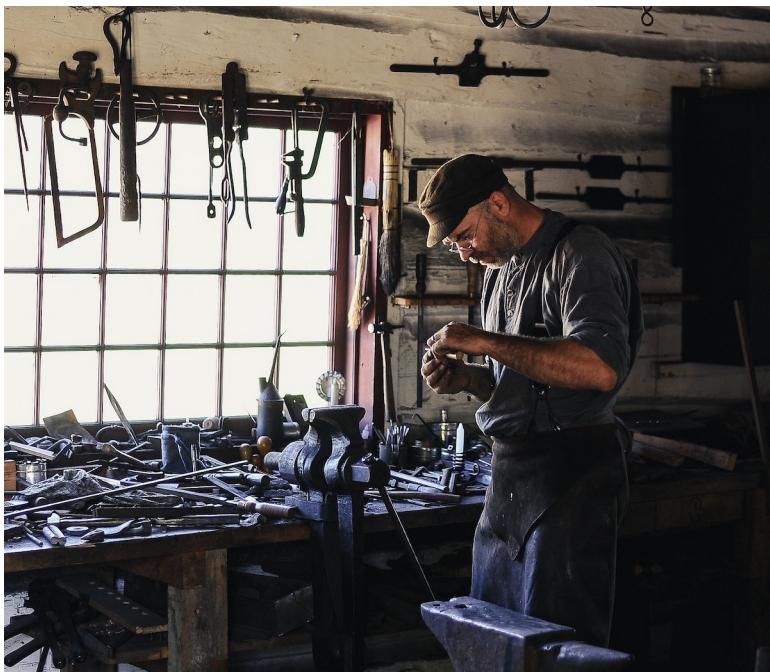


image by pexels from pixabay



**Architecture Dashboard Examples:** [source code of examples<sup>1</sup>](#) of how to build an Architecture Data Dashboard as a part of the Grounded Architecture Data Foundation. The dashboard is a simple static website generated from JSON files and [published via GitHub pages<sup>2</sup>](#).

---



**Sokrates<sup>3</sup>:** Sokrates is a tool I built to implement my vision of documenting and analyzing software architectures of complex systems. Sokrates provides a pragmatic, inexpensive way to extract rich data from source code repositories. Sokrates can help you understand your code by making visible the size, complexity, and coupling of software and all people interactions and team topologies.

---

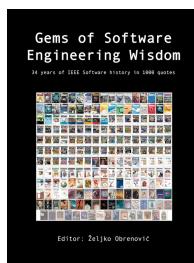
<sup>1</sup><https://github.com/zeljkoobrenovic/grounded-architecture-dashboard-examples>

<sup>2</sup><https://zeljkoobrenovic.github.io/grounded-architecture-dashboard-examples/>

<sup>3</sup><https://sokrates.dev>



**Productivity Tools**<sup>4</sup>: a collection of more than 100 online tools I built to help me in my daily work as an architect. I reuse these tools and lessons learned in building these tools when designing Data Foundation parts of the Grounded Architecture.



**obren359.com**<sup>5</sup>: I've created a curated collections and high-quality IT knowledge resources (articles, videos, podcasts).

---

<sup>4</sup><https://obren.io/tools>

<sup>5</sup><https://obren359.com>

## 27: Favorite Quotes

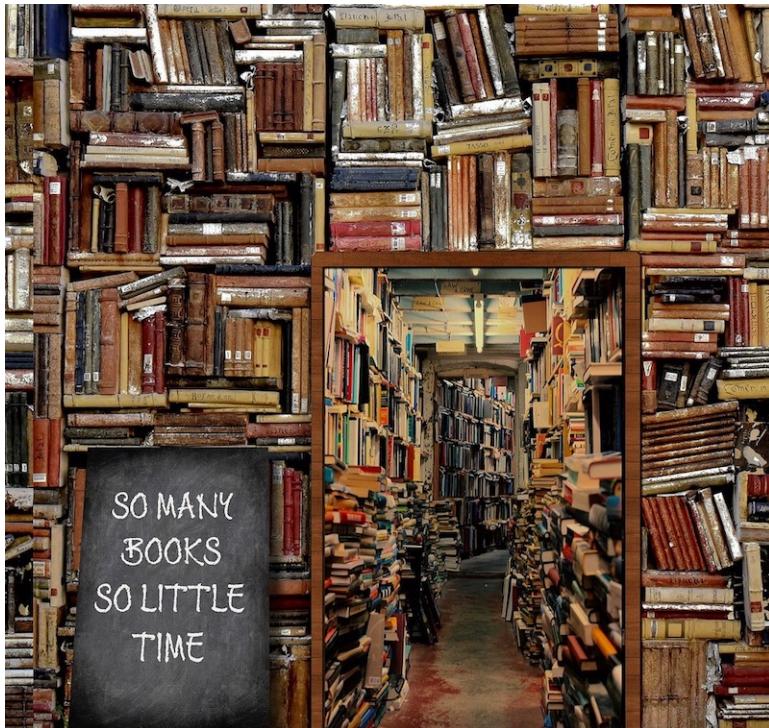


image by amy from pixabay

## 27.1: People, People, People

1

*Architecture is **not so much about the software, but about the people** who write the software. The core principles of architecture, such as **coupling and cohesion**, aren't about the code. The code doesn't 'care' about how cohesive or decoupled it is; ... But people do care about their coupling to other team members.* -James O. Coplien

---

*Software design is **an exercise in human relationships**.* -Kent Beck

---

*Bad software architecture is a people problem. When **people don't work well together they make bad decisions**.* -Kate Matsudaira

---

*To change the architecture of a software-intensive system ensconced in a large organization, you often have to change the architecture of the organization. And ultimately, that is **a political problem, not just a technical one**.* -Grady Booch

## 27.2: Changing Role of Architecture

1

*The role of architects has changed from trying to be the smartest person to **making everyone else smarter**.* -Gregor Hohpe

---

*The architect's role is changing from being primarily a decision maker to being a coordinator, advisor, and knowledge manager ... **a central knowledge hub.*** -Rainer Weinreich & Iris Groher

---

*Rather than inject itself in every decision loop, **architecture should design mechanisms for teams to make better decisions** - they'll know better anyhow, and slowing them down carries a high cost..* -Gregor Hohpe

---

*Your architecture team's job is to **solve your biggest problems**. The best setup is the one that allows it to accomplish that.* -Gregor Hohpe

---

*Your organization has to **earn its way to an effective architecture function**. You can't just plug some architects into the current mess and expect it to solve all your problems.* -Gregor Hophe

## 27.3: Complexity

1

*Excessive complexity is nature's punishment for organizations that are unable to make decisions.* -Gregor Hohpe

---

*Metawork is more interesting than work. ... we love complicated little puzzles to solve, so we keep overengineering everything.* -Neal Ford

---

*Developers are drawn to complexity like moths to a flame often with the same result.* -Neal Ford

## 27.4: Other

1

*Architecture is just a collective hunch, a **shared hallucination**, an assertion by a set of stakeholders on the nature of their observable world, be it a world that is or a world as they wish it to be.* -Grady Booch

---

***Architects code**, but not to deliver code. Rather, to understand the ramifications of the decisions that they're making.* -Gregor Hohpe, Dave Farley

---

*Architectural decisions are design decisions that are **hard to make** or **costly to change**.* -Olaf Zimmermann

# **Part VI: Appendix**

## 28: Appendix Overview



image by pexels from pixabay

**IN THIS SECTION, YOU WILL:** Get an overview of three resources in the appendix.

As we bring this book to a close, I would like to equip you with a helpful appendix. In this part, I outline three resources that have proven vital in my journey as an IT architect and have reminded me about what I always need to know about crucial aspects of IT practice. I routinely refer back to them and carry them in my practitioner “backpack,” so I added their summaries in the appendix.

- **EIC/ISO 25010 Standard** focuses on product quality and system quality models. While imperfect, this standard is a reasonably complete yet compact source for understanding software maintainability, security, reliability, and performance efficiency.
- **Cloud Design Patterns** offer a mix of crucial distributed system and messaging system topics combined with modern public cloud engineering themes.
- **Characteristics of High Performing Organizations** from the ‘Accelerate’ book by Nicole Forsgren, Jez Humble, and Gene Kim is an excellent source of empirical knowledge about crucial practices of high-performing technology organizations.

## 29: ISO 25010 Standard



image by pexels from pixabay

**IN THIS SECTION, YOU WILL:** Get an overview of ISO/IEC 25010 standard that provides guidelines and recommendations for evaluating software product quality.

## 29.1: Overview

ISO/IEC 25010 is a standard that provides guidelines and recommendations for evaluating software product quality. It is a part of the ISO/IEC 25000 series, which encompasses a set of international standards for software engineering.

The standard identifies several quality characteristics that should be considered during evaluation. I have used four of them in practice and can recommend them as a pragmatic way to assess and discuss the quality of software systems:

- **Maintainability:** The ease with which the software can be modified, corrected, adapted, or enhanced.
- **Security:** The software's ability to protect information and data from unauthorized access, disclosure, alteration, destruction, or disruption.
- **Performance Efficiency:** The software's ability to perform its functions efficiently, considering resource utilization.
- **Reliability:** The software's ability to maintain a specified level of performance under stated conditions for a defined period.

## 29.2: Maintainability

**Definition:** The degree of effectiveness and efficiency with which the intended maintainers can modify a product or system.

### 29.2.1: Maintainability Characteristics

- **Analyzability:** The degree of effectiveness and efficiency with which it is possible to assess the impact on a product or system of an intended change to one or more of its parts or diagnose a product for deficiencies or causes of failures, or identify parts to be modified.
- **Modifiability:** The degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality.
- **Testability:** The degree of effectiveness and efficiency with which test criteria can be established for a system, product, or component, and tests can be performed to determine whether those criteria have been met.
- **Modularity:** The degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.
- **Reusability:** The degree to which an asset can be used in multiple systems or building other assets.

### 29.2.2: Maintainability Tactics

- **Keep Systems Small:** The overall size of the source code of the software product matters. Systems with about 200,000 lines of code are frequently challenging to maintain.
- **Organize System in Limited Number (e.g., 7±2) Balanced Components:** When a system has too many or too few

components, or if components differ in size significantly (e.g., you have 80% of code in one common component), it is considered more challenging to understand and maintain.

- **Couple Files Loosely:** Software products where more source code resides in files strongly coupled with other files are deemed more challenging to maintain.
- **Keep Components Loosely Coupled:** Hide most files in a component from direct dependencies from other components.
- **Avoid Duplication:** Avoid the occurrence of identical fragments of source code in more than one place in the product.
- **Write Short Units of Code:** Keep units (e.g., functions, methods) small, with many practitioners recommending units to be shorter than 30 lines. Short units are also much easier to unit test.
- **Write Simple Units of Code:** Keep the number of decision points (so-called McCabe index, e.g., if, while statement) per unit low, ideally below 5.
- **Avoid Many Parameters in Unit Interfaces:** A large number of parameters in a unit are deemed more complicated to maintain.

## 29.3: Security

The degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization.

### 29.3.1: Security Characteristics

- **Confidentiality:** Ensures that data are accessible only to those authorized.
- **Integrity:** Prevents unauthorized access or modifications.
- **Non-repudiation:** actions or events can be proven to have occurred.
- **Accountability:** Actions of an entity can be traced uniquely to the entity.
- **Authenticity:** The identity of a subject or resource can be proved to be the one claimed.

### 29.3.2: Security Tactics

- **Protect Data Transport:** Protect data transport with a sufficiently robust protection method, minimizing caching of sensitive data.
- **Protect Stored Data:** Prevent or restrict access to data stored physically. Encrypt sensitive data. Correctly applying a one-way hash to the data.
- **Verify Input and Output:** Reject invalid system input. The rejection should not disclose information. Validate within the system, validated against a whitelist. Prevent injection and overflow vulnerabilities. Escape all output. Never unnecessarily expose implementation details.

- **Uniquely Identify Actors:** Identify and record system actors in a way that points uniquely to a specific actor. Include detailed traceable information, such as location or origin.
- **Enforce In-Depth Authentication:** Enforce authentication for all system functions and all uses. Use an intrinsically robust authentication method. For failed authentication, do not perform any tasks or expose information.
- **Enforce In-Depth Authorization:** Authorize within the system so the user cannot circumvent it. Authorize for every system function and at every attempt. If authorization fails, record this event and inform the user only that authorization failed. Give users the least possible privileges.
- **Manage User Sessions Securely:** Create and expire sessions and tokens securely.
- **Keep Evidence:** Allow non-repudiation and accountability. Keep the proof that an actor actively approved and performed an action. Store it securely, and facilitate retrieval and analysis.
- **Manage Users Securely:** Implement a secure and automated process for user sign-up, blocking and removal, and management of user credentials.

## 29.4: Performance Efficiency

**Definition:** Performance relative to the resources used under stated conditions.

### 29.4.1: Performance Efficiency

- **Time Behavior:** The degree to which a product or system's response, processing times, and throughput rates meet requirements when performing its functions.
- **Capacity:** The degree to which the maximum limits of a product or system parameter meet requirements.
- **Resource Utilization:** The degree to which the amounts and types of resources a product or system use meet requirements when performing its functions.

### 29.4.2: Performance Efficiency Tactics

- **Observe System Performance:** Know the system's actual performance and support problem analysis and resolution by measuring and monitoring the system.
- **Optimize Internal Communication:** Limit the number of steps, usage of small messages, and transformations in inter-process communications.
- **Limit External Communication:** Limit usage of external services and associated uncertainty, especially if the system has no strong guarantees over the external service's time behavior.
- **Optimize Common Transactions:** Identify the most common and critical transactions, and apply standard strategies for their optimization.

- **Scale Transaction Capabilities:** Make it easy to increase transaction processing capacities when needed, both for individual components and the whole system.
- **Scale Data Capabilities:** Take care of the volume and characteristics of the data.
- **Isolate External Influences:** Control or exclude external influences that may impact the performance and the consistency of response times.
- **Provision Resources Elastically:** Accommodate variations in workload. so that the consumed computer resources and associated costs of a service rise and fall proportionally to the workload.

## 29.5: Reliability

The degree to which a system, product, or component performs specified functions under specified conditions for a specified period.

### 29.5.1: Reliability Dimensions

- **Maturity:** The system is thoroughly tested and has a low manual maintenance effort, minimizing the number of potential errors in production.
- **Availability:** The system is thoroughly tested and has a low manual maintenance effort, minimizing the number of potential errors in production.
- **Fault-Tolerance:** Fitted with mechanisms to ensure a certain error tolerance level, ensuring that not every error results in a system failure.
- **Recoverability:** Should the system fail despite all efforts, it has mechanisms to either recover fully automatically or support human intervention for fast recovery.

### 29.5.2: Reliability Tactics

- **Isolate Faults:** Prevent faults propagating from one to other components.
- **Prevent Inconsistent States (Transaction Handling):** Prevents inconsistent states and data in the presence of errors.
- **Avoid Single Points of Failure (Redundancy):** Redundancy determines to which extent a system is vulnerable to a single point of failure.
- **Automate Deployment:** The faster a system can be (re)deployed, the faster new versions can be put into

production, enabling more rapid recovery from errors and failures.

- **Make System Autonomous:** Avoid the dependency of a system on human intervention to stay operational.
- **Test Reliability:** Make test loads resemble production loads.
- **Implement Failover:** Make switching to another component easy when one fails.

# 30: Cloud Design Patterns



image by donna kirby from pixabay

**IN THIS SECTION, YOU WILL:** Get a mix of key distributed and messaging system topics combined with modern public cloud engineering themes.

## 30.1: Overview

Cloud Design Patterns offer a mix of key distributed and messaging system topics and modern public cloud engineering themes.

I grouped these patterns into the following categories:

- Performance and Scalability,
- Resiliency,
- Messaging,
- Management and Monitoring,
- Security, and
- Other.

Source: [learn.microsoft.com/en-us/azure/architecture/patterns](https://learn.microsoft.com/en-us/azure/architecture/patterns)<sup>1</sup>

---

<sup>1</sup><https://learn.microsoft.com/en-us/azure/architecture/patterns/>

## 30.2: Performance and Scalability

Make your system perform well under load and react well to the load change.

- **Command and Query Responsibility Segregation (CQRS):** Segregate operations that read data from operations that update data by using separate interfaces.
- **Event Sourcing:** Use an append-only store to record the entire series of events that describe actions taken on data in a domain.
- **Materialized View:** Generate prepopulated views over the data in one or more data stores when the data is not ideally formatted for required query operations.
- **Index Table:** Create indexes over the fields in data stores frequently referenced by queries.
- **Sharding:** Divide a data store into horizontal partitions or shards.
- **Static Content Hosting:** Deploy static content to a cloud-based storage service that can deliver them directly to the client.
- **Cache-Aside:** Load data on demand into a cache from a data store.
- **Throttling:** Control the consumption of resources used by an instance of an application, an individual tenant, or an entire service.
- **Rate Limit Pattern:** Limiting pattern to help you avoid or minimize throttling errors related to these throttling limits and to help you more accurately predict throughput.
- **Geodes:** Deploy a collection of backend services into geographical nodes, each of which can service any request for any client in any region.

### 30.3: Resiliency

Gracefully handle and recover from failures.

- **Bulkhead:** Isolate elements of an application into pools so that if one fails, the others will continue to function.
- **Retry:** Enable an application to handle anticipated, temporary failures when connecting to a service or network resource by transparently retrying a previously failed operation.
- **Circuit Breaker:** Handle faults that might take a variable amount of time to fix when connecting to a remote service or resource.
- **Compensating Transaction:** Undo the work performed by a series of steps, which together define an eventually consistent operation.
- **Leader Election:** Coordinate the actions performed by a collection of collaborating task instances in a distributed application by electing one instance as the leader responsible for managing the other instances.

## 30.4: Messaging

Create a messaging infrastructure that connects the components and services, ideally loosely coupled, to maximize scalability.

- **Publisher/Subscriber:** Enable an application to announce events to multiple interested consumers asynchronously without coupling the senders to the receivers.
- **Competing Consumers:** Enable multiple concurrent consumers to process messages received on the same messaging channel.
- **Pipes and Filters:** Break down a task that performs complex processing into a series of separate elements that can be reused.
- **Priority Queue:** Prioritize requests sent to services so that requests with a higher priority are received and processed more quickly than those with a lower priority.
- **Queue-Based Load Leveling:** Use a queue that acts as a buffer between a task and a service that it invokes to smooth intermittent heavy loads.
- **Scheduler Agent Supervisor:** Coordinate actions across a distributed set of services and other remote resources.
- **Asynchronous Request-Reply:** Decouple backend processing from a frontend host, where backend processing needs to be asynchronous, but the frontend still needs a clear response.
- **Choreography:** Have each system component participate in the decision-making process about the workflow of a business transaction instead of relying on a central point of control.
- **Saga:** Manage data consistency across microservices in distributed transaction scenarios. A saga is a sequence of transactions that updates each service and publishes a message or event to trigger the next transaction step.
- **Sequential Convoy:** Process a set of related messages in a defined order without blocking the processing of other groups of messages.

## 30.5: Management and Monitoring

Expose runtime information that administrators and operators can use to manage and monitor the system—supporting changing business requirements and customization without requiring the application to be stopped or redeployed.

- **Health Endpoint Monitoring:** Implement functional checks in an application that external tools can access through exposed endpoints at regular intervals.
- **Ambassador:** Create helper services that send network requests on behalf of a consumer service or application.
- **Anti-Corruption Layer:** Implement a façade or adapter layer between a modern application and a legacy system.
- **External Configuration Store:** Move configuration information from the application deployment package to a centralized location.
- **Gateway Aggregation:** Use a gateway to aggregate multiple individual requests into a single request.
- **Gateway Offloading:** Offload shared or specialized service functionality to a gateway proxy.
- **Gateway Routing:** Route requests to multiple services using a single endpoint.
- **Sidecar:** Deploy components of an application into a separate process or container to provide isolation and encapsulation.
- **Strangler Fig:** Incrementally migrate a legacy system by gradually replacing specific pieces of functionality with new applications and services.

## 30.6: Security

Prevent malicious or accidental actions outside of the designed usage, and prevent disclosure or loss of information.

- **Federated Identity:** Delegate authentication to an external identity provider.
- **Gatekeeper:** Protect applications and services using a dedicated host instance that is a broker between clients and the application or service, validates and sanitizes requests, and passes requests and data between them.
- **Valet Key:** Use a token or key that provides clients restricted direct access to a specific resource or service.
- **Claim Check:** Split a large message into a claim check and a payload.

## 30.7: Other Patterns

Create good designs. Take care of consistency, coherence, maintainability, and reusability.

- **Compute Resource Consolidation:** Consolidate multiple tasks or operations into a single computational unit.
- **Backends for Frontends:** Create different backend services to be consumed by specific frontend applications or interfaces.
- **Deployment Stamps:** Deploy multiple independent copies of application components, including data stores.

# 31: High Performing Technology Organizations

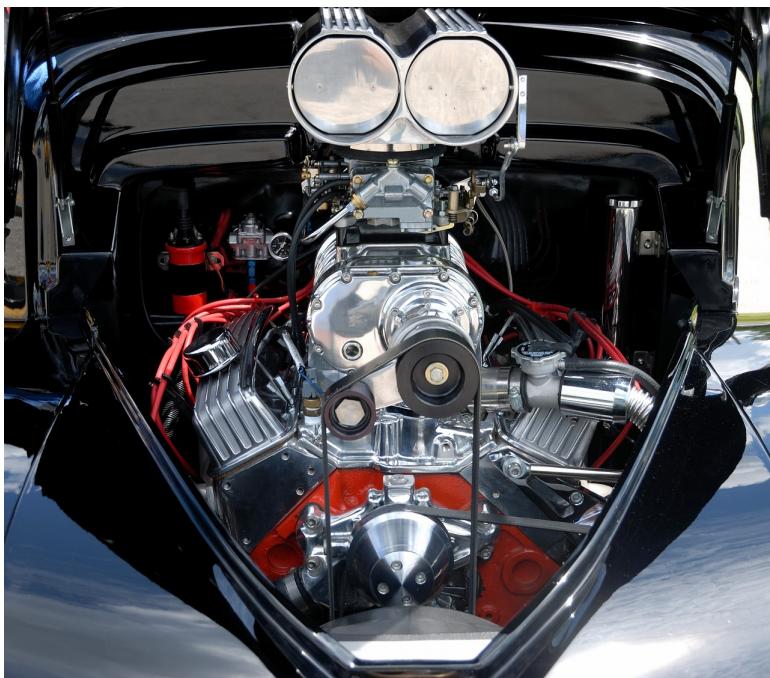


image by paul brennan from pixabay

**IN THIS SECTION, YOU WILL:** Get a summary of the characteristics of high-performing technology organizations.

## 31.1: Overview

Characteristics of High Performing Organizations from the [Accelerate book](#)<sup>1</sup> by Nicole Forsgren, Jez Humble, and Gene Kim is an excellent source of empirical knowledge about high-performing IT organizations.

I summarized several critical insights from this book:

- Overview of **four key metrics**, describing metrics that provide suitable measures of organization performance.
- Overview of **critical practices of high-performing technology organizations** grouped into the following categories: Continuous Delivery, Architecture, Product and Process, Lean Management and Monitoring Capabilities, Culture

---

<sup>1</sup><https://www.oreilly.com/library/view/accelerate/9781457191435/>

## 31.2: Four Key Metrics

- **Lead time for changes:** Elite performers have a lead time for changes of less than 1 hour (from code committed to code successfully running in production).
- **Deployment frequency:** Elite performers have a deployment frequency of multiple times daily.
- **Time to restore service:** Elite performers have a mean time to recover (MTTR) of less than 1 hour.
- **Change failure rate:** Elite performers have a 0-15% change failure rate.

## 31.3: Practices

### 31.3.1: Continuous Delivery

- **Use version control for all production artifacts:** For all production artifacts, including application code, application configurations, system configurations, and scripts for automating the build and configuration of the environment.
- **Automate your deployment process:** The degree to which deployments are fully automated and do not require manual intervention.
- **Implement continuous integration:** Code is regularly checked in, and each check-in triggers a set of quick tests to discover serious regressions, which developers fix immediately.
- **Use trunk-based development methods:** Fewer than three active branches; branches and forks having very short lifetimes (e.g., less than a day); teams rarely or never having “code lock” periods.
- **Implement test automation:** Software tests are run automatically (not manually) continuously through the development process.
- **Support test data management:** Test data requires careful maintenance, and test data management is becoming an increasingly important part of automated testing.
- **Shift left on security:** Integrating security into the design and testing phases of the software development process is vital to driving IT performance.
- **Implement continuous delivery (CD):** Software is in a deployable state throughout its lifecycle, and the team prioritizes keeping the software in a deployable state over working on new features.

### 31.3.2: Architecture

- **Use a loosely coupled architecture:** The extent to which a team can test and deploy their applications on demand without requiring orchestration with other services.
- **Architect for empowered teams:** Teams that can choose which tools to use, do better, and, in turn, drive better software development and delivery performance.

### 31.3.3: Product and Process

- **Gather and implement customer feedback:** Actively and regularly seek customer feedback and incorporate this feedback into the design of products.
- **Make the flow of work visible through the value stream:** Teams should have a good understanding of and visibility into the flow from the business to customers, including the status of products and features.
- **Working in small batches:** Teams should slice work into small pieces that can be completed in a week or less.
- **Foster and enable team experimentation:** Team experimentation is the ability of developers to try out new ideas and create and update specifications during the development process without requiring approval from outside of the team, which allows them to innovate quickly and create value.

### 31.3.4: Lean Management and Monitoring Capabilities

- **Have a lightweight change approval process:** A lightweight change approval process based on peer review (pair programming or intrateam code review) produces

superior IT performance than using external change approval boards (CABs).

- **Monitor across applications and infrastructure to inform business decisions:** Use data from application and infrastructure monitoring tools to take action and make business decisions. This monitoring goes beyond paging people when things go wrong.
- **Check system health proactively:** Monitor system health using threshold and rate-of-change warnings to enable teams to detect and mitigate problems preemptively.
- **Improve process and manage work with work-in-progress (WIP) limits:** Using work-in-progress limits to manage work flow is well known in the Lean community. When used effectively, this drives process improvement, increases throughput and makes constraints visible in the system.
- **Visualize work to monitor quality and communicate throughout the team:** Visual displays, such as dashboards or internal websites, used to monitor quality and work in progress have contributed to software delivery performance.

### 31.3.5: Culture

- **Support a generative culture (as outlined by Westrum):** Hallmarks of this measure include good information sharing, high cooperation and trust, bridging between teams, and conscious inquiry.
- **Encourage and support learning:** Is learning, in your culture, considered essential for continued progress? Is learning thought of as a cost or an investment?
- **Support and facilitate collaboration among teams:** Reflects how well traditionally siloed teams interact in development, operations, and information security.

- **Provide resources and tools that make work meaningful:** This measure of job satisfaction is about doing challenging and meaningful work and being empowered to exercise your skills and judgment.
- **Support or embody transformational leadership:** Comprised of five factors: vision, intellectual stimulation, inspirational communication, supportive leadership, and personal recognition.