



# 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 available at <http://leanpub.com/groundedarchitecture>

This version was published on 2024-11-11



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>	1
1.1: Grounded Architecture Overview	3
1.2: What Will You Learn?	6
1.3: Beyond Rigid Frameworks: IT Architecture As an Adaptable Outcome-Oriented Practice	9
1.4: Key Influences	12
1.5: Why This Book?	13
1.6: A Part of the Bigger Picture: A Trilogy in Four Parts	14
1.7: A Bit of Personal History	16
1.8: The Structure of the Book	17
1.9: Stay Connected	18
1.10: Acknowledgments	19
<b>2: Context: Fast-Moving Global Organizations</b>	20
2.1: Global Scale	22
2.2: Multi-Dimensional Diversity	24
2.3: Nonlinear Growth Dynamics	27
2.4: Synergy and Transformation Pressures	29
2.5: Decentralization and Loose Coupling	31
2.6: Questions to Consider	33
<b>3: Goals: Adapting, Growing, and Using Data</b>	34
3.1: Goal 1: Executing At Scale	36
3.2: Goal 2: Adaptivity	37
3.3: Goal 3: Increasing Quality of Decisions with Data	38
3.4: Goal 4: Maximizing Organizational Alignment	39
3.5: Goal 5: Maximizing Organizational Learning	40
3.6: Questions to Consider	41

## CONTENTS

<b>Part I: Grounded Architecture Framework . . . . .</b>	<b>42</b>
<b>4: Grounded Architecture Framework: Introduction . . . . .</b>	<b>43</b>
<b>5: Lightweight Architectural Analytics . . . . .</b>	<b>47</b>
5.1: Examples of Lightweight Architectural Analytics Tools . . . . .	50
5.2: Requirements For A Lightweight Architectural Analytics . . . . .	57
5.3: Building Lightweight Architectural Analytics . . . . .	59
5.4: Using Lightweight Architectural Analytics . . . . .	61
5.5: To Probe Further . . . . .	62
5.6: Questions to Consider . . . . .	63
<b>6: Collaborative Networks . . . . .</b>	<b>64</b>
6.1: Background: Central vs. Federated Architecture Practice . . . . .	67
6.2: The Hybrid Model . . . . .	69
6.3: Building Collaborative Networks . . . . .	73
6.4: To Probe Further . . . . .	75
6.5: Questions to Consider . . . . .	76
<b>7: Operating Model: General Principles . . . . .</b>	<b>77</b>
7.1: Examples of Architecture Activities . . . . .	80
7.2: Guiding Principles for Architectural Excellence: Policies, Autonomy, and Engagement . . . . .	83
7.3: Embracing Diversity . . . . .	90
7.4: Setting Boundaries . . . . .	91
7.5: To Probe Further . . . . .	94
7.6: Questions to Consider . . . . .	95
<b>8: Cooperation-Based Operating Model: Six Simple Rules . . . . .</b>	<b>96</b>
8.1: Background: Hard and Soft Management . . . . .	98
8.2: Collaboration Approach . . . . .	101
8.3: Rule 1: Understand What Your People Do . . . . .	103
8.4: Rule 2: Reinforce Integrators . . . . .	105
8.5: Rule 3: Increase the Total Quantity of Power . . . . .	107
8.6: Rule 4: Increase Reciprocity . . . . .	109
8.7: Rule 5: Extend the Shadow of the Future . . . . .	111
8.8: Rule 6: Reward Those Who Cooperate . . . . .	113
8.9: To Probe Further . . . . .	115
8.10: Questions to Consider . . . . .	116

## CONTENTS

<b>9: Operating Model: Nudge, Taxation, Mandates . . . . .</b>	<b>117</b>
9.1: Nudging . . . . .	120
9.2: Taxation (Economic Incentives) . . . . .	122
9.3: Mandates and Bans . . . . .	124
9.4: Questions to Consider . . . . .	126
<b>10: Transforming Organizations with Grounded Architecture . . . . .</b>	<b>127</b>
10.1: Executing at Scale . . . . .	129
10.2: Adaptivity . . . . .	132
10.3: Improving the Quality of Decision-Making with Data . . . . .	135
10.4: Maximizing Organizational Alignment . . . . .	138
10.5: Maximizing Organizational Learning . . . . .	140
10.6: Questions to Consider . . . . .	143
<b>Part II: On Being Architect . . . . .</b>	<b>144</b>
<b>11: On Being Architect: Introduction . . . . .</b>	<b>145</b>
11.1: Growing As An Architect . . . . .	148
11.2: Thinking Like an Architect . . . . .	151
<b>12: Building Skills . . . . .</b>	<b>152</b>
12.1: Technical Skills . . . . .	156
12.2: Soft Skills . . . . .	159
12.3: Product Development Skills . . . . .	162
12.4: Business Skills . . . . .	164
12.5: Decision-Making Skills . . . . .	167
12.6: Integrating Skills for Success . . . . .	169
12.7: To Probe Further . . . . .	171
12.8: Questions to Consider . . . . .	172
<b>13: Making Impact . . . . .</b>	<b>173</b>
13.1: Pillars of Impact . . . . .	176
13.2: Questions to Consider . . . . .	185
<b>14: Leadership Traits . . . . .</b>	<b>186</b>
14.1: David Marquet's Work: The Leader-Leader Model . . . . .	189
14.2: Netflix's Valued Behaviors: Leadership Behaviors . . . . .	193
14.3: Questions to Consider . . . . .	198

## CONTENTS

<b>15: Thinking Like an Architect: Architects as Superglue . . . . .</b>	<b>199</b>
15.1: Superglue Architects . . . . .	201
15.2: Superglueing in Action #1: Aligning Organization . . . . .	203
15.3: Supergluing In Action #2: Aligning Discussions About Problems, Solutions and Implementations . . . . .	210
15.4: Superglue Impact: Keeping Everyone in the Same Boat, Upon a Stormy Sea . . . . .	215
15.5: To Probe Further . . . . .	217
15.6: Questions to Consider . . . . .	218
<b>16: Thinking Like an Architect: Balancing Curiosity, Doubt, Vision, and Skepticism . . . . .</b>	<b>219</b>
16.1: Curiosity and Wonder . . . . .	223
16.2: Doubt . . . . .	227
16.3: Vision and Belief . . . . .	230
16.4: Skepticism . . . . .	233
16.5: Balancing Motivators in IT Architecture . . . . .	237
16.6: To Probe Further . . . . .	239
16.7: Questions to Consider . . . . .	240
<b>17: Architects' Career Paths . . . . .</b>	<b>241</b>
17.1: Solid Engineering Background . . . . .	243
17.2: Entering Architecture Space . . . . .	245
17.3: Career Progression in IT Architecture . . . . .	247
17.4: Career Progression Beyond IT Architecture . . . . .	249
17.5: To Probe Further . . . . .	251
17.6: Questions to Consider . . . . .	252
<b>Part III: On Human Complexity . . . . .</b>	<b>253</b>
<b>18: On Human Complexity: Introduction . . . . .</b>	<b>254</b>
<b>19: The Culture Map: Architects' Culture Compass . . . . .</b>	<b>256</b>
19.1: Communicating . . . . .	259
19.2: Evaluating . . . . .	263
19.3: Persuading . . . . .	267
19.4: Leading . . . . .	271
19.5: Deciding . . . . .	275
19.6: Trusting . . . . .	279

## CONTENTS

19.7: Disagreeing . . . . .	283
19.8: Scheduling . . . . .	287
19.9: Rules . . . . .	291
19.10: Questions to Consider . . . . .	292
<b>20: The Human Side of Decision-Making . . . . .</b>	<b>293</b>
20.1: Understanding Human Biases and Limitations . . . . .	295
20.2: Understanding Power and Limitations of Human Intuition in Decision-Making . . . . .	306
20.3: Understanding Human Indecisiveness . . . . .	309
20.4: Understanding Group Dynamics in Decision-Making . . . . .	318
20.5: Questions to Consider . . . . .	323
<b>21: Effortless Architecture . . . . .</b>	<b>324</b>
21.1: IT Doesn't Have To Be As Hard and Complicated As We Make It . . . . .	327
21.2: Effortless State . . . . .	329
21.3: Effortless Action . . . . .	339
21.4: Effortless Results . . . . .	350
21.5: The Road to Effortless Achievement . . . . .	363
21.6: To Probe Further . . . . .	364
21.7: Questions to Consider . . . . .	365
<b>Part IV: Expanding the Architect's Toolkit . . . . .</b>	<b>366</b>
<b>22: Expanding the Architect's Toolkit: Learning From Other Fields . . . . .</b>	<b>367</b>
<b>23: Architecture in Product-Led Organizations: Learning From Customer-Centric Fields . . . . .</b>	<b>370</b>
23.1: The Build Trap . . . . .	373
23.2: The Discipline of Market Leader . . . . .	380
23.3: Product Operations . . . . .	383
23.4: Questions to Consider . . . . .	387
<b>24: Decision Intelligence in IT Architecture: Learning From Data, Social, and Managerial Fields . . . . .</b>	<b>388</b>
24.1: Basics of Decision-Making . . . . .	391

24.2: Preparing for Making Decisions . . . . .	394
24.3: Decision-Making Complexity . . . . .	399
24.4: Decision is a Step in the Process . . . . .	403
24.5: Decision-Making With Data and Tools . . . . .	406
24.6: To Probe Further . . . . .	409
24.7: Questions to Consider . . . . .	410
<b>25: Economic Modeling With ROI and Financial Options:</b>	
<b>Learning From the Finance Field . . . . .</b>	<b>411</b>
25.1: The Return-on-Investment Metaphor . . . . .	414
25.2: The Financial Options Metaphor . . . . .	416
25.3: Communication Frameworks . . . . .	418
25.4: To Probe Further . . . . .	423
25.5: Questions to Consider . . . . .	424
<b>26: How Big Transformations Get Done: Learning From Mega-Projects . . . . .</b>	<b>425</b>
26.1: The Iron Law of Mega-Projects . . . . .	428
26.2: Heuristics for Successful Mega-Projects . . . . .	446
26.3: To Probe Further . . . . .	470
26.4: Questions to Consider . . . . .	471
<b>Part V: Summary . . . . .</b>	<b>472</b>
<b>27: Summary . . . . .</b>	<b>473</b>
<b>28: To Probe Further: Online Appendix Overview . . . . .</b>	<b>478</b>

# 1: Introduction



image by rasica from istock

**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 an architecture practice to all organizational levels as an antidote to the “ivory tower” architecture.
- I also explain my motivation to write this book.

Have you ever wondered how to run an IT architecture practice without **feeling stuck in an ivory tower**? Look no further! This book will share an approach to running an IT architecture practice in larger organizations based on my experience at AVIV Group, eBay Classifieds, and Adevinta. I call it “Grounded Architecture”—because it’s all about having your feet on the ground and your architecture **firmly planted in reality**.

The approach presented in this book connects architecture to every nook and cranny of your organization, making sure you’re not a lone wizard in a tower casting spells that no one understands. Grounded Architecture prioritizes connections with people and data over fancy theories, processes, and tools. The books will offer you new insight into IT architecture, redefining it as an **adaptable** and **outcome-oriented** practice.

The name “Grounded Architecture” is a hat-tip to the **Grounded Theory**<sup>1</sup> methodology. Grounded Theory is about developing theories from the data you gather from the real world rather than pulling them out of thin air. Likewise, Grounded Architecture evolves from honest feedback and data, not from dusty old books of abstract principles.

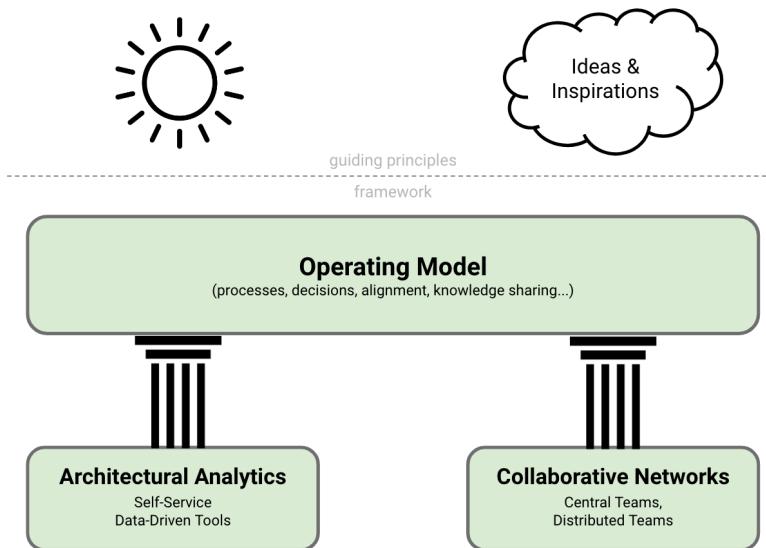
---

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

## 1.1: Grounded Architecture Overview

This book breaks down Grounded Architecture into two main parts (Figure 1):

- **Framework:** The nuts and bolts you need to get your Grounded Architecture practice up and running.
- **Guiding Principles:** Handy tips and inspirations to help you bring these ideas to life.



*Figure 1: Grounded Architecture Overview.*

Grounded Architecture framework consists of three elements:

- **Lightweight Architectural Analytics,**
- **Collaborative Networks,**
- **Operating Model.**

*Lightweight Architectural Analytics* keeps your architecture decisions based on up-to-date and complete information about your organization's tech landscape.

*Collaborative Networks* represents a strong network of people working on architecture across your organization, ensuring your efforts make a difference.

Lastly, the sections *Operating Model: General Principles*, *Cooperation-Based Operating Model*, and *IT Governance: Nudge, Taxation, Mandates* define a collection of processes and agreements that lets architects do their thing, leveraging data and people connections to create a powerful, organization-wide impact.

As part of my Grounded Architecture quest, I've devised several guiding principles and tools to help you sprinkle Grounded Architecture magic in your practice. While traditional IT architecture literature often zeros in on technical know-how, this collection dares you to think differently about IT architecture and to dive into the rich insights of social, behavioral, and management sciences. These resources are grouped into several parts and are sure to make your journey both insightful and entertaining:

- **On Being an Architect:**

- Building Skills
- Making Impact
- Leadership
- Thinking Like an Architect: Architects as Superglue
- Thinking Like an Architect: Balancing Curiosity, Doubt, Vision, and Skepticism
- Architects' Career Paths

- **On Human Complexity:**

- The Culture Map: Architects' Culture Compass
- The Human Side of Decision-Making
- Effortless Architecture

- **Expanding the Architect's Toolkit: Learning From Other Fields:**

- Architecture in Product-Led Organizations: Learning From Customer-Centric Fields
- Decision Intelligence in IT Architecture: Learning From Data, Social, and Managerial Fields
- Economic Modeling With ROI and Financial Options: Learning From the Finance Field

- How Big Transformations Get Done: Learning From Mega-Projects

If all this is not enough, in the online appendix I outline several resources that have proven vital in my journey as an IT architect. I routinely refer back to them and carry them in my practitioner “backpack,” so I added their summaries in the online [appendix](#), including:

- Favorite Quotes<sup>2</sup>
- Bookshelf<sup>3</sup>
- Resources for Managing, Growing, and Hiring Architects<sup>4</sup>
- Resources for Effective Communication<sup>5</sup>
- Resources for Working With Toxic Colleagues<sup>6</sup>
- Resources for Dealing With Scapegoating at Work<sup>7</sup>
- EIC/ISO 25010 Standard<sup>8</sup>
- Cloud Design Patterns<sup>9</sup>

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

---

<sup>2</sup><https://grounded-architecture.io/quotes>

<sup>3</sup><https://grounded-architecture.io/bookshelf>

<sup>4</sup><https://grounded-architecture.io/career-resources>

<sup>5</sup><https://grounded-architecture.io/communication>

<sup>6</sup><https://grounded-architecture.io/toxic-colleagues>

<sup>7</sup><https://grounded-architecture.io/scapegoating>

<sup>8</sup><https://grounded-architecture.io/iso25010>

<sup>9</sup><https://grounded-architecture.io/cloud-design-patterns>

## 1.2: What Will You Learn?

This book serves as a guideline to build and operate a **robust IT architecture practice**, ensuring alignment with organizational goals while adapting to the challenges of modern IT environments.

### 1.2.1: Key Topics

The book will provide you with tips and inspirations about work that you need to set up and run a modern IT architecture practice, including:

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

I also provide a number of other ad-hoc lessons and inspirations from my work.

### 1.2.2: 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.

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.

### 1.2.3: 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, but there are already many great resources. This book is about **expanding your horizons** to apply your technical skills in complex organizations. You can broaden your horizons as a head or manager of architects and organize and support architects to use their technical skills more effectively as a team.

### 1.2.4: 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, 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.

### 1.2.5: Who Should Read This Book?

When writing this book, I had a broad audience in mind. The articles should be helpful to both technical and non-technical people. The book can help IT architects 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.

### 1.2.6: Applying Ideas In Practice

I approach designing an architecture practice in complex organizations as an **art of cooking**. A modern IT architecture practice is like a master

chef's kitchen. As you will always work with many local ingredients, you can't just copy a recipe from one restaurant to another and expect it to taste the same.

This book shares some tasty tips and savory secrets on "cooking" up a thriving architecture practice. But remember, **every kitchen is different**. Some ingredients and recipes are essential, while others might be left on the shelf. And don't be surprised if you need to add a pinch of something special from other sources to spice things up.



image by hispanolistic from istock

When I start an architecture practice, I see myself as a chef stepping into a new restaurant with a suitcase full of favorite spices. I blend in core preferred elements and foundational frameworks like a chef using their trusty spices and kitchen tools. But the **authentic flavor**—the magic that makes a dish unforgettable—comes from the **local ingredients**. In an organization, these local ingredients are the people: the in-house talent and culture that give the enterprise its unique taste.

While fundamental structures and practices may be consistent across diverse businesses, the most crucial elements—akin to fresh, **local ingredients** in cooking—you must procure locally. The staff's skills, experiences, and insights are the secret ingredients that personalize and refine the architecture to align with specific business needs and goals. They are what render the architecture truly unique and effective.

## 1.3: Beyond Rigid Frameworks: IT Architecture As an Adaptable Outcome-Oriented Practice

In this book, I will use the terms **IT architecture** and **IT architects** in a broad sense. I don't adhere to strict divisions between enterprise, solution, technical, or software architecture, nor do I narrowly define roles such as enterprise architects, solution architects, or Staff+ technical leaders like principal engineers. Instead, I view IT architecture as a **cohesive function**—a bridge connecting technology, business, product, and organizational strategy. IT architecture is more than a set of duties; it serves as essential connective tissue, aligning these domains to work harmoniously.

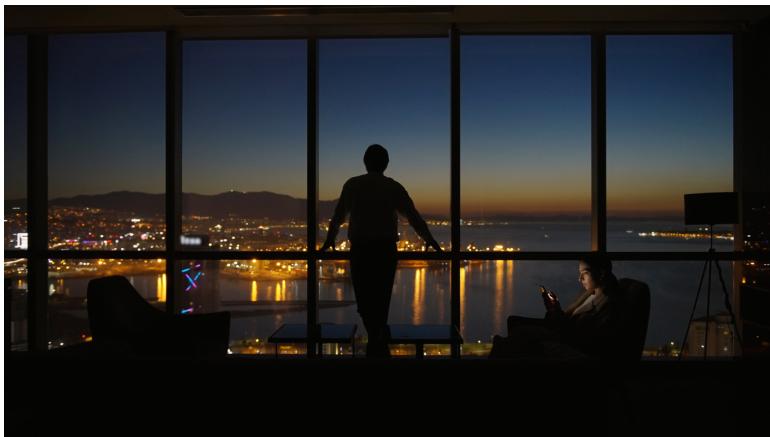


image by brightstars from istock

### 1.3.1: Grounding and Empowerment

Rather than advocating for rigid role distinctions, I propose **rethinking and empowering these roles** with an **outcome-oriented, data-informed perspective**, where each role is adaptable and focused on delivering value and impactful outcomes that align with the organization's broader goals. This book emphasizes **grounding** your architecture in the organization with data, people connections, and pragmatic operating

principles. You do not necessarily need to change your architecture team or roles to benefit from this grounding.

I believe IT architecture **isn't one-size-fits-all**; it adapts to each company's context, spanning strategic guidance, system integration, solution design, technical implementation, or hands-on leadership. This **adaptability** makes architecture both **powerful and relevant**, enabling architects to align with business goals while remaining responsive to technological change.

In some organizations, architecture may emphasize **business alignment**, ensuring technology investments support the mission. In others, it might focus on **technical depth**, where architects closely collaborate with engineering to build robust, scalable systems. Despite these variations, the core function remains: architecture guides coherence in complex ecosystems. For this reason, I consider IT architects anyone performing this bridging work, regardless of their title.

Focusing on architecture as an **integrative and adaptable** function allows us to appreciate the unique contributions of each architectural role. Rather than defining roles rigidly, this approach highlights architecture as a continuous, evolving practice that balances immediate technical needs with **long-term strategic vision**. This perspective empowers architecture to drive sustainable growth, adaptability, and innovation across the business.

### 1.3.2: Accepting Challenges

This broader view of IT architecture, while robust and integrative, presents unique challenges. Unlike traditional frameworks with well-defined roles and responsibilities, my perspective **requires constant adaptation** to evolving organizational needs. Traditional frameworks bring clarity and stability, outlining roles and providing structured pathways, making it easier to align efforts across a project's lifecycle.

In contrast, an adaptive, integrative view of architecture is **less rigid** and **more attuned** to a company's specific, shifting needs. This flexibility allows architects to bridge technology, business, product, and strategy holistically, responsive to change. However, IT architects and technical leaders must continually **evaluate and recalibrate** their approach. This evolution might mean adjusting priorities, balancing high-level strategy

with hands-on guidance, and frequently engaging with stakeholders to ensure alignment.

I believe deeply in this approach, though I acknowledge its demands. It calls for **insight** into business and technology, a readiness to **embrace ambiguity**, and the ability to make decisions amid shifting priorities. IT architects working within this model need a strong grasp of the company's vision and an **acute sense of timing**—knowing when to emphasize strategy or technical details. It requires agility, resilience, and the willingness to pivot as needs shift and to challenge assumptions to serve the business.

While challenging, this adaptive model of architecture offers rewards that traditional frameworks may lack. By not confining architects to strict roles, it enables solutions genuinely aligned with the organization's mission, fostering closer collaboration and responsiveness to new opportunities or challenges. This flexibility ultimately allows architecture to drive lasting value and growth—creating a foundation that supports both current goals and future innovation.

## 1.4: Key Influences

The Grounded Architecture approach also builds on many ideas that others have successfully used. Gregor Hohpe's **Architecture Elevator**<sup>10</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>11</sup> and Will Larson's book **Staff Engineer: Leadership beyond the management track**<sup>12</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. You can find them in the online [Bookshelf appendix](#)<sup>13</sup>.

---

<sup>10</sup><https://architectelevator.com/>

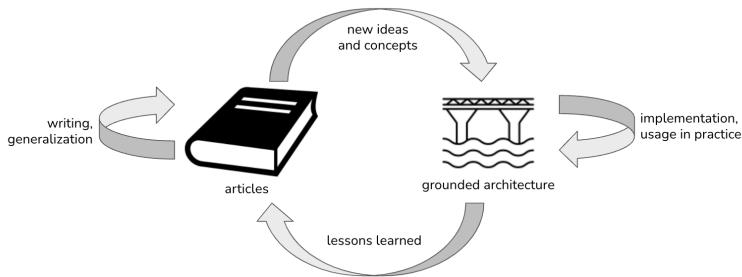
<sup>11</sup><https://www.oreilly.com/library/view/the-staff-engineers/9781098118723/>

<sup>12</sup><https://staffeng.com/guides/staff-archetypes/>

<sup>13</sup><https://grounded-architecture.io/bookshelf>

## 1.5: Why This Book?

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



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

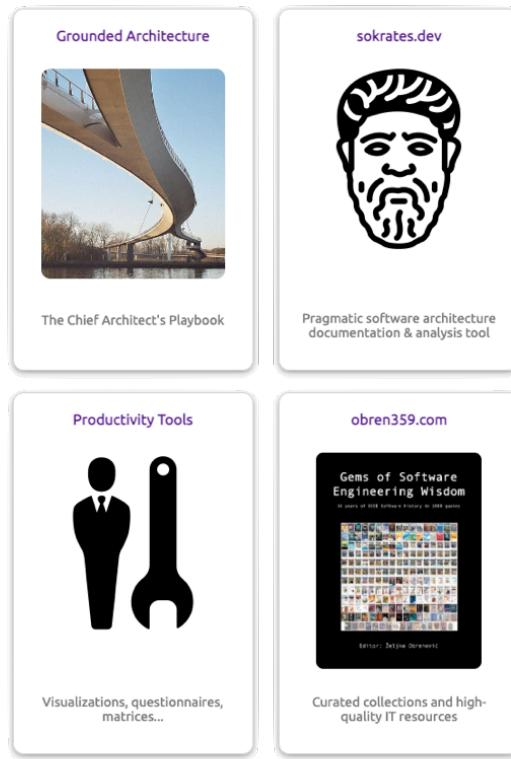
I also needed this book for the **education of architects** and to **increase awareness about modern architecture practices** in organizations I worked in. Having written content can significantly help to spread the message. As nicely described by Hohpe written word has distinct advantages over the spoken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

## 1.6: 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>14</sup> I have built 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.

The other resources include:

<sup>14</sup><https://obren.io/>

- **Sokrates**<sup>15</sup>: an open-source polyglot source 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>16</sup>: A collection of more than 100 online tools I built to help me in my daily work.
- **359° Overview of Tech Trends**<sup>17</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>18</sup>.

---

<sup>15</sup><https://sokrates.dev>

<sup>16</sup><https://obren.io/tools>

<sup>17</sup><https://www.obren359.com/>

<sup>18</sup><https://obren.io/>

## 1.7: 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>19</sup>, where 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>20</sup>, which enables efficient and pragmatic extraction of data about technology and organization from source code. This work has directly influenced my view of **Lightweight Architectural Analytics** part of the Grounded Architecture framework.

My experience as a **CTO** of **Incision**<sup>21</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>22</sup> and **Eindhoven Technical University (TU/e)**<sup>23</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>24</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, a vital part of IT architects' job.

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

---

<sup>19</sup><https://www.softwareimprovementgroup.com/>

<sup>20</sup><https://sokrates.dev>

<sup>21</sup><https://incision.care>

<sup>22</sup><https://www.cwi.nl/en/>

<sup>23</sup><https://www.tue.nl/en/>

<sup>24</sup><https://design-instability.com/>

## 1.8: 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 framework and describe its three elements: Lightweight Architectural Analytics, Collaborative Networks, and the Operating Model.

In the third part, I discuss the Guiding Principles of Grounded Architecture, grouped into several sub-parts: On Being an Architect, On Soft Skills, On Decision-Making, On Human Complexity, and On Execution and Governance.

Finally, I conclude with a summary and encourage you to explore further with the help of the plentiful external resources I've provided.

## 1.9: Stay Connected

You can find additional resources online at:

- <https://grounded-architecture.io><sup>25</sup>

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

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

---

<sup>25</sup><https://grounded-architecture.io/>

## 1.10: Acknowledgments

I want to 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 the development of data-informed architecture in our organizations.

The cover image is [a photo of Nesciobrug<sup>26</sup>](#). Credit: the botster, CC BY-SA 2.0, via Wikimedia Commons.



image by henk monster cc by 3.0 via wikimedia commons

---

<sup>26</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 work on creating and running an architecture practice isn't just a lofty idea; it's a **practical approach** sharpened from **real-world experience**. My perspective comes from lessons I learned as the Chief Architect at AVIV Group and the Principal Architect at eBay Classifieds and Adevinta.

To better grasp presented ideas or solutions, it is helpful to understand the problems we were trying to solve and **the context** in which these ideas were born. Here's a peek into the context that shaped my Grounded Architecture approach:

- **Global scale:** the organizations I worked in were operating across multiple countries and continents with millions of users.
- **Multidimensional diversity:** these organizations were diverse in terms of their customer base, workforce, business models, team topologies, and technology stacks.
- **Nonlinear growth dynamics:** in addition to organic growth, complex organizations change their portfolio through mergers and acquisitions of new businesses or divestments.
- **Synergies and transformation pressures:** complex organizations want to exploit the benefits of economies of scale and reduce duplication of efforts.
- **Decentralized, loosely coupled organizational units:** organizational units have significant autonomy while working together on common goals.

## 2.1: Global Scale

I have honed my approach within genuinely global and multicultural organizations on a massive scale:

- Operating across numerous **geographies, cultures, and languages**,
- Serving **millions of users** daily,
- Collaborating with thousands of software **developers** across hundreds of product and development **teams**,
- Implementing systems comprising hundreds of **millions of lines of source code**.



image by pete linforth from pixabay

Operating on a global scale introduces several compelling opportunities for organizations. It can significantly increase organizational effectiveness by **reducing duplication of effort** through centralized shared activities. Additionally, leveraging **economies of scale** allows for cost advantages, such as lowering the unit prices of utilized technologies. Global operations also enhance **business resilience and flexibility**, enabling compensation for local market fluctuations with global resources. The expansive talent pool available to global organizations supports local and international initiatives. Moreover, these organizations possess

significant resources to invest in supporting nonlinear growth through mergers and acquisitions (M&As).

However, the global and massive scale also presents numerous challenges. It results in **high organizational complexity**, with thousands of potential communication channels within the organization. The **complex technology landscape** entails numerous interconnected services. Managing a large talent pool incurs **high workforce costs**. Furthermore, such organizations face high computing resource expenses due to the need to serve a vast customer base around the clock. The operational complexity increases with high and variable customer demands across multiple locations. Additionally, global organizations have a **vast attack surface**, with many potential entry points for attackers. Lastly, any manual process, such as creating an organizational or technology landscape overview, is limited due to the scale involved.

Balancing opportunities and challenges on a global scale has been one of the most demanding and rewarding aspects of my architectural work. Such a magnitude makes any manual process inefficient and difficult to scale. The global scale was one of the main drivers behind the aggressive datafication of our architecture practice. It has also led us to create more decentralized collaborative networks and operating models to execute and track decision-making across the board.

## 2.2: Multi-Dimensional Diversity

The organizations I worked with were incredibly diverse across multiple dimensions:

- **Cultures:** A varied workforce and clientele, both local and remote.
- **Organization:** Units of different sizes, complexities, and organizational styles.
- **Product:** Diverse product features catering to various markets and customer segments.
- **IT Architecture:** Combination of legacy systems and modern approaches.
- **Technology:** Numerous programming languages and thousands of third-party libraries, frameworks, and services.



image by simon from pixabay

For instance, I worked with organizational units differing in several aspects, including **unit size**, which ranged from hundreds of employees

to just a dozen. The **team topologies** varied, spanning from single-team setups to hierarchical team organizations. Additionally, the **architectural roles** varied, with some units having dedicated local architecture teams and lead architects. In contrast, in smaller units, team members handled architectural duties alongside other responsibilities.

Similarly, technology-wise, we managed a range of styles in active production systems, from legacy **monolithic** applications to intricate modern **microservice** and **serverless** ecosystems. Each organizational segment had its own unique history and legacy systems. Our technology stack was extensive, covering multiple mainstream technologies. The infrastructure included several public cloud providers such as AWS, GCP, Azure, and custom-built private data centers. Our systems employ various application technologies, including database technologies like MySQL, PostgreSQL, MongoDB, Cassandra, AWS RDS, and more. The backend programming languages used were Java, C#, Go, Scala, PHP, Node.js. We used Swift, Objective-C, Java, Kotlin, Flutter/Dart, and more for mobile app programming. The frontend programming languages and frameworks included React, AngularJS, Vue, jQuery, and others.

Diversity offers several **opportunities**, including increased technology **innovation**. A diverse workforce can explore a variety of technologies and tools creatively. It also leads to better implementation because access to a broader **pool of diverse resources** allows for the selection of the best tool for the job.

However, diversity also brings challenges. One such challenge is **increased complexity**, resulting in a higher system landscape complexity and greater cognitive load for teams mastering numerous topics simultaneously. Additionally, there is **reduced flexibility**, as expertise spread across many domains and technologies limits reorganization possibilities. Furthermore, diversity can lead to higher **technical debt** due to multiple technology stacks, increasing legacy components, and outdated technologies. While diversity is a rich source of new possibilities from an architectural perspective, it always necessitates carefully managing complexity.

Diversity has influenced our an architecture practice in multiple ways. It has led to the development of lean tools that can cover a broad range of technology stacks rather than the adoption of specialized ones that can go deep on one stack but cannot cover 95% of our landscape. We also adopted a more flexible governance model to help all teams in a practical way that

is aligned with their diverse ways of working.

## 2.3: Nonlinear Growth Dynamics

Complex organizations like the ones I have worked in are often highly dynamic. These organizations frequently undergo significant growth, contraction, and reorganization, evolving both **organically** and **inorganically**.



image by pixels from pixabay

**Organic growth** refers to internal expansion driven by the company's own operations. **Inorganic change** involves acquiring other businesses, opening new locations, or divesting parts of the company.

Nonlinear growth, in particular, can be advantageous in several scenarios. It can **rapidly increase the customer base** or introduce new market segments. Additionally, such changes can **accelerate innovation** by incorporating new technologies or services.

However, nonlinear growth dynamics significantly impact architectural activities. The sudden integration of new companies **increases organizational complexity**, introducing many new units. Acquiring a new company also **brings in new technology and engineering units**, along with their unique processes and technology stacks. Furthermore, these nonlinear dynamics **necessitate a flexible architecture** to accommodate potential divestitures.

Nonlinear growth offers substantial benefits but also challenges managing increased complexity and the need to maintain architectural flexibility.

In terms of its impact on an architecture practice, such dynamics lead to constantly high levels of complexity and more uncertainty. This has led us to prioritize the creation of better transparency to track changes that such dynamics introduce. We also needed to collaborate closely with business and finance stakeholders on developing tools for economics and risk modeling of investments and divestments.

## 2.4: Synergy and Transformation Pressures

Complex organizations aim to grow not just in size, but also in efficiency by leveraging economies of scale, cost synergies, and enhancing their capacity for innovation. Our investors expect us to become **more than the sum of our original parts.**



image by mustangjoe from pixabay

Pursuing synergies and transformations offers several opportunities, such as **cost reductions** through less duplication and lower expenses. **Accelerated innovation** can occur as savings from cost reductions free up resources for new developments. Additionally, creating synergistic components enables more possibilities for **reuse and sharing**, while well-executed transformations result in increased efficiency and **lower unit costs**.

However, striving for synergies and efficiency presents challenges. There is a need for significant **initial investment** to realize benefits, which carries high risks. Performance pressure arises as teams must deliver excellent **short-term results** while undergoing significant transformations.

Balancing transformation activities with regular work can temporarily **reduce productivity**. Moreover, post-transformation, the organization and technology landscape may become more complex due to increased dependencies, such as reusing central services.

The pressure to achieve synergies and efficiency can lead to high expectations and complicate regular architectural work. Nonetheless, these forces also create numerous opportunities for growth and improvement. For an architecture practice, these pressures created a strong need for better tracking project costs, value, and risks. Being able to calculate and back with data decisions for both innovative projects and legacy retirements was a critical aspect of our work.

## 2.5: Decentralization and Loose Coupling

Researcher Karl Weick developed the concepts of tight and loose coupling to describe organizational structures, initially 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 such organizations, management can directly coordinate different departments' activities according to a central strategy.

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



image by andrii yalanskyi from istock

Most organizational units I worked with were loosely coupled. Our companies frequently grew through acquisitions of companies in different marketplaces. Business strategies also promoted the independent evolution of local units to address local market needs more effectively and quickly. These units often enjoyed a high level of autonomy, frequently with their development teams and sometimes with local CFOs, CMOs, or CEOs.

Loose coupling offers several advantages. It provides **higher flexibility**, allowing units to develop independently and address specific needs without synchronizing with other units. This flexibility leads to **reduced time-**

**to-market**, as fewer dependencies enable marketplaces to rapidly change and evolve their products for local needs. Additionally, loose coupling **fosters innovation** by offering opportunities to quickly explore ideas in smaller contexts.

However, loose coupling also presents several challenges. It can lead to duplication of effort, as local market needs might differ but often have significant overlap in product features and technology, resulting in **redundant efforts** as each marketplace creates solutions for the same problems. This approach also increases **accidental diversity**, where limited synchronization may result in significantly different design and technology choices for the same problem, making it difficult to consolidate solutions, move personnel between teams, or benefit from economies of scale. Moreover, loose coupling results in **limited control**, as fewer dependencies and varying goals make it more challenging to implement changes across the organization.

From an architectural perspective, loose coupling presents an interesting challenge, often leading to a conflict between global alignment and control and local autonomy. For our architecture practice, decentralization and loose coupling led to many changes. We emphasized “hands off, eyes on,” leaving teams autonomy in their work but creating complete transparency based on data. Our operating model has a high level of decentralization to enable both scaling and alignment of architecture work.

## 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 your architecture practice?*
- *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: Adapting, Growing, and Using Data**



image by bluehouse skis from pixabay

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

**KEY POINTS:**

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

Grounded Architecture emerged as a necessity in response to our **intricate and multifaceted challenges**. The Grounded Architecture framework was designed to address these specific challenges. By moving away from manual processes and embracing automation, data-driven decision-making, and adaptive frameworks, we aimed to create a **more resilient and effective** an architecture practice.

In following sections I will outline a breakdown of the goals I set for an architecture practice:

- Goal 1: Executing At Scale,
- Goal 2: Adaptivity,
- Goal 3: Increasing Quality of Decisions with Data,
- Goal 4: Maximizing Organizational Alignment,
- Goal 5: Maximizing Organizational Learning.

### 3.1: Goal 1: Executing At Scale

Our organizations were like a bustling city with hundreds of teams and thousands of projects, each with its own unique complexity and requirements. Traditional, one-size-fits-all approaches to an architecture practice simply couldn't keep up with this dynamic environment. We needed a system that could support this vast and varied ecosystem. Grounded Architecture was designed to **handle such diversity at scale**, ensuring that teams and projects received the tailored support they needed without being bogged down by rigid processes.

Some of the success criteria for this goal included:

- Always having the **full transparency** about the technology landscape. Without full transparency, it isn't easy to understand the landscape's complexity or your work's context. This transparency should include good data and visualization of the size and quality of all source code repositories, public cloud accounts and technologies, private data centers, development efforts, etc.
- Having mechanisms and spaces to maintain **practical working relationships** with all development teams and key stakeholders. Knowing the organizational landscape and having spaces for engagement is crucial.
- Being able to scale and grow the organization **without** introducing significant **slowing of decision-making**. Finding the right balance between teams' autonomy and alignment is essential.

## 3.2: Goal 2: Adaptivity

In our dynamic environments, significant change is not just frequent; it's expected. Whether these changes are organic, like evolving business needs, or inorganic, like mergers and acquisitions, our architecture must be able to adapt swiftly. Grounded Architecture was crafted to be flexible and responsive, allowing us to pivot quickly in response to new challenges and opportunities. This adaptability ensures that our architectural framework remains relevant and effective, no matter how the organizational landscape shifts.

Some of the success criteria for this goal included:

- Being able to track and support **legacy and new technologies**, adapting this support as the organization transforms, grows, acquires new legacy, and adopts new technologies.
- Having **readily available data** for analyses of different business scenarios (e.g., retire legacy vs. investment in legacy, buy-or-build, divestments).
- Being able to **routinely onboard** and quickly understand the technology landscape of acquired companies.

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

Relying on gut feelings or individual opinions is always insufficient and risky when dealing with operations at scale. Decisions need to be based on solid data to **ensure accuracy and reliability**. Grounded Architecture aims to incorporate tools and mechanisms to support data-driven decision-making. By leveraging data and analytics, we can move away from subjective opinions and towards more objective, evidence-based decisions. This approach should enhance the quality of our decisions and facilitate their consistency and alignment with our organizational goals.

Some of the success criteria for this goal included:

- Always having **complete, up-to-date data** about all key elements of the organizational technology landscape.
- Having technical data connected with **product** and **business data** (e.g., vibrancy vs. public cloud costs).
- Making data available via **self-service tools** for the organization so that more people can make data-informed decisions.
- Ensuring **routine usage** of the data in decision-making.

### 3.4: Goal 4: Maximizing Organizational Alignment

In a global, fast-moving organization, misalignment can quickly become the norm. Different teams and departments might pursue conflicting objectives, leading to inefficiencies and confusion. Grounded Architecture aims to **serve as a cohesive force**, promoting alignment across the entire organization. Providing a clear, unified framework helps to minimize misalignments. It facilitates all parts of the organization working towards common goals. This alignment is crucial for maintaining efficiency and avoiding the chaos that can arise from disparate efforts.

Some of the success criteria for this goal included:

- Having pragmatic **standardized guidelines** and best practices (e.g., golden paths) for technology use, ensuring consistency across the organization.
- Fostering a **culture of collaboration** and knowledge sharing among teams to align on technology choices and implementation strategies.
- Regularly **reviewing and updating** technology standards to align with evolving business needs and industry trends.
- Identifying and eliminating **redundant processes** and activities to streamline operations and reduce wasted resources.

## 3.5: Goal 5: Maximizing Organizational Learning

Staying current with emerging technologies and industry trends is essential for maintaining a competitive edge. Still, it can be challenging when dealing with the demands of legacy systems. Grounded Architecture should facilitate continuous learning and growth. It supports the rapid adoption of new technologies and encourages **ongoing education and training**. Grounded Architecture should ensure we always have the best tools and knowledge to drive innovation and improvement.

Some of the success criteria for this goal included:

- Organizing frequent workshops, seminars, and training sessions to facilitate **sharing knowledge** and best practices across the organization.
- Encouraging employees at all levels to contribute to and participate in knowledge-sharing initiatives, promoting a **culture of continuous learning**.
- Developing platforms and tools that enable **easy access** to shared knowledge and resources, enhancing collective expertise.
- Involving **diverse stakeholders**, including developers, managers, and end-users, in discussions and decision-making processes related to architecture and technology.
- **Creating opportunities** for employees from various departments to engage in architectural planning and feedback sessions to learn from each other.

These criteria aim to create an inclusive and dynamic environment where knowledge is freely shared, and diverse contributions are not just welcomed, but valued.

## 3.6: Questions to Consider

Knowing what goals an architecture practice needs to support in your organization is crucial to defining structures and measuring 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 an architecture practice? Does your scale require special measures to ensure your an 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 an architecture practice 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 Framework**

## 4: Grounded Architecture Framework: Introduction

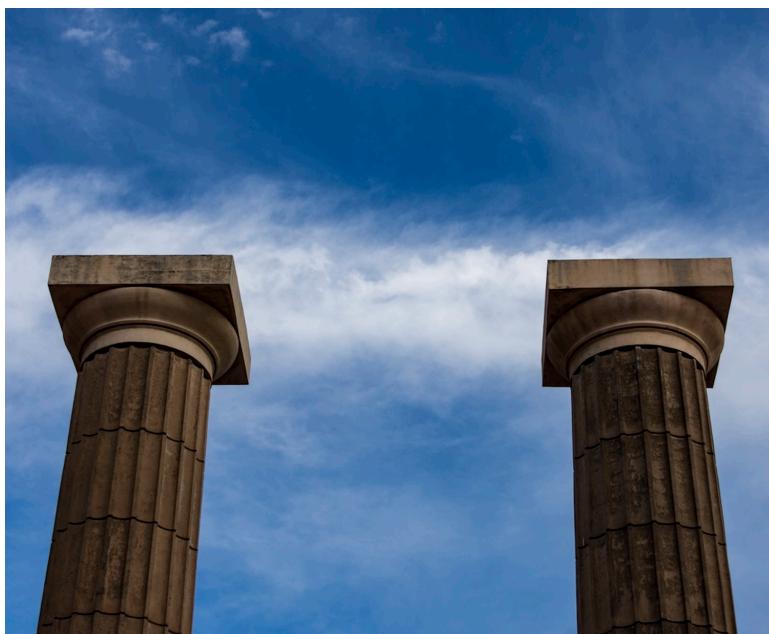


image by dario truoc from istock

**IN THIS SECTION, YOU WILL:** Get an overview of the Grounded Architecture framework: Lightweight Architectural Analytics, Collaborative Networks, and Operating Model.

**KEY POINTS:**

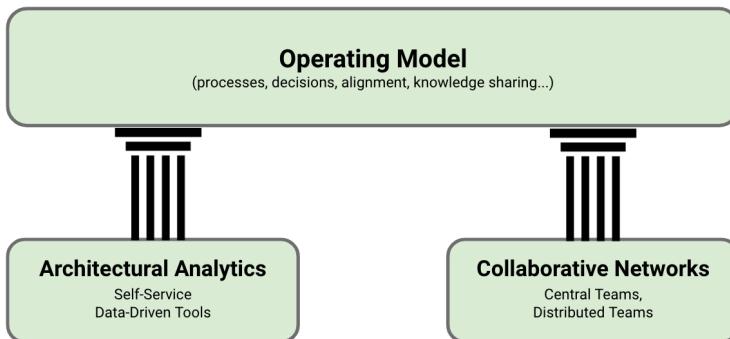
- I introduce three elements of Grounded Architecture Framework: Lightweight Architectural Analytics, Collaborative Networks, and The Operating Model as an approach to setting organizational structures for a modern IT architecture practice.
- Prioritizing people interactions and data over processes and tools, Grounded Architecture aims to connect an architecture practice to all organizational levels as an antidote to the “ivory tower” architecture.

In this part, I will briefly introduce the Grounded Architecture framework. I chose the term “Grounded Architecture” to highlight that the primary goal of my approach is **avoid having an “ivory tower” architecture practice** disconnected from the organization. This disconnection is a real danger in a **fast-moving, global, and diverse setting**.

The Grounded Architecture framework is designed to be a **practical solution**. It aims to be **deeply rooted** in the organization, prioritizing **people’s interactions** and **data** over processes and tools. The goal is to connect an architecture practice to all parts and levels of the organization, serving as a practical antidote to the ‘ivory tower’ architecture. In this part of the ebook, I will delve into the framework’s structure and how it can be applied in real-world scenarios.

Grounded Architecture framework is an approach to setting organizational structures for an architecture practice, and it has three elements:

- Lightweight Architectural Analytics,
- Collaborative Networks,
- The Operating Model.



*Figure 1: The Grounded Architecture framework.*

*Lightweight Architectural Analytics* 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. [Lightweight Architectural Analytics section](#) provides more details.

*Collaborative Networks* connect all the people doing architecture across the organization. These networks are crucial to ensure that an architecture practice has any **tangible impact**. The [Collaborative Networks section](#) provides more details.

Lastly, the *Operating Model* defines:

- **General Principles:** I've defined several **general guidelines** on directing architects to do everything an architecture practice typically does, leveraging Lightweight Architectural Analytics and Collaborative Networks to create a data-informed, organization-wide impact.
- **Cooperation-Based Operating Model: Six Simple Rules:** Grounded Architecture is particularly well-suited for organizations organized based on cooperation. As businesses grow increasingly complex, the importance of cooperation within organizational structures cannot be overstated. The "Six Simple Rules" framework emphasizes that successful organizations leverage their members' collective judgment and cooperation. By fostering a culture of collaboration, organizations can better utilize their capabilities to address and resolve complex problems.

- **IT Governance: Nudge, Taxation, Mandates:** Governance in technology is a critical aspect of any operating model. I propose a governance model incorporating three styles: nudging, taxation, and mandates. Nudging involves subtle policy shifts that guide behavior without restricting choices, while taxation uses economic incentives to encourage desired outcomes. Mandates and bans provide direct regulations to ensure compliance and standardization. This multifaceted approach allows for flexible and effective governance, balancing innovation with control and ensuring technological initiatives align with broader organizational goals.

While I elaborately discuss many faces of the operating model, I want to emphasize that any Operating Model is only relevant if there are **healthy Lightweight Architectural Analytics and Collaborative Networks**. Without data and people connections, an operating model leads to an ivory tower institution, generating opinion-based decisions disconnected from reality.

Now that we've completed the high-level Grounded Architecture tour let's examine the specifics.

## 5: Lightweight Architectural Analytics



image by ko\_orn from istock

**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:**

- Lightweight Architectural Analytics serves as a medium to create a complete, up-to-date picture of critical elements of the organization's technology landscapes.
- Such analytics 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 Lightweight Architectural Analytics, I have been creating open-source tools that can help you obtain valuable architectural insights from data sources, such as source code repositories. Check out open-source [architecture dashboard examples<sup>1</sup>](#) and [Sokrates<sup>2</sup>](#).

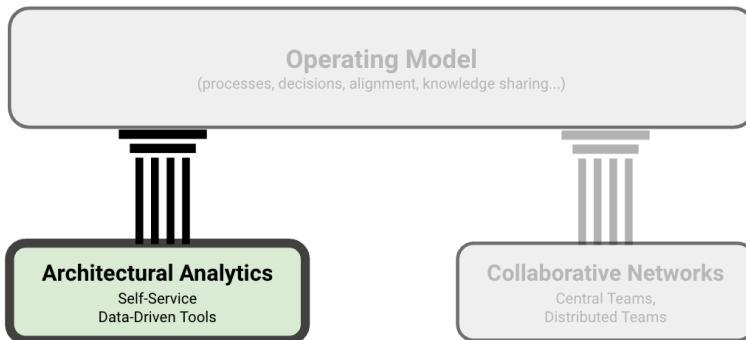
*"If we have data, let's look at data. If all we have are opinions, let's go with mine."* — Jim Barksdale

Everywhere I worked on creating an architecture practice, I strongly (aka obsessively) **emphasized data**. Consequently, one of the first steps I make in any an architecture practice is to create an Lightweight Architectural Analytics to get a complete, up-to-date picture of critical elements of an organization's technology landscapes (Figure 1). Manual documentation does not scale, and relying on data ensures the reliability and scalability of decision-making.

---

<sup>1</sup><https://zeljkoobrenovic.github.io/grounded-architecture-dashboard-examples/>

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



*Figure 1: The Grounded Architecture framework: Lightweight Architectural Analytics.*

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

I call the tools I will discuss in this section **lightweight** to emphasize that, you don't need to begin with costly or overly complicated software platforms. Plenty of simple and affordable tools—many of them open-source—can help you gain meaningful insights without breaking the bank.

## 5.1: Examples of Lightweight Architectural Analytics Tools

To illustrate what I mean by Lightweight Architectural Analytics, I will give a few concrete examples from my recent work (Figure 2).

The dashboard is titled "aviv group" and "Architecture Team Dashboard". Below the title is the tagline "Reducing Subjectivity in Architecture with Data and Insights".

**source code analyses**

- House of CARDs**: source: source code analysis. Icon: A stack of cards.
- All Repositories**: source: source code analysis. Icon: A magnifying glass over a code editor.
- GitLab Repositories**: source: source code analysis. Icon: A stylized mountain or pyramid shape.
- Bitbucket Repositories**: source: source code analysis. Icon: A bucket icon.
- Other Repositories**: source: source code analysis. Icon: Three dots followed by an ellipsis.

**operations**

- AWS Usage Explorer**: source: cloud billing report. Icon: AWS logo.
- GCP Usage Explorer**: source: cloud billing report. Icon: Cloud icon.
- Data Center Migration**: source: migration inventory. Icon: Server rack icon.
- Incidents Explorer**: source: slack. Icon: Flame icon.
- Standard Data Models**: source: data modeling coe. Icon: Database icon.

**transformation**

- Whitelabel Capabilities**: source: capabilities sheet. Icon: Elephant icon.
- Product Portfolio**: source: product portfolio slides. Icon: Hands holding a cube icon.
- Legacy Commitment**: source: source code. Icon: Person standing next to a sphere icon.
- Real Estate Ecosystem**: source: product team. Icon: Circular nodes icon.

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

I typically implement Lightweight Architectural Analytics as a dashboard with many data apps, typically leveraging the following data sources:

- **Source code** contains an incredible amount of information about

technology, people's activity, team dependencies, and the quality of software systems. By analyzing commit histories, code complexity, and contributions, you can identify critical areas of improvement, understand team dynamics, and ensure code quality.

- **Public cloud billing reports** provide an overview of trends in used cloud services, regions, and budgets. Monitoring billing reports can help manage budgets, identify cost-saving opportunities, and understand usage patterns across different services and regions.
- **Incident reports** can reveal trends and dependencies among incidents. Analyzing these reports can reveal trends, common issues, and dependencies among incidents, helping manage problems and improving system reliability.
- **Key business metrics**, like vibrancy, can show user activity on our systems. Tracking these metrics can help assess the business's health, understand user behavior, and guide strategic decisions to enhance user experience.
- **Activity reports from messaging and collaboration platforms (such as Slack)** can help understand discussion topics and team interactions. Analyzing these reports can help understand collaboration patterns, identify key discussion areas, and improve team communication and productivity.

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

### 5.1.1: Example 1: Source Code and Commit History Analytics

The **source code** and its **commit history** are like a treasure chest for creating data-driven architecture documentation—packed with nuggets of wisdom about **technology**, **team activities**, **dependencies**, and software quality.

Many tools can help you reveal these insights. In this section, I will focus on a simple tool I built to get these insights. To help you dig up this treasure without getting your hands too dirty, I've developed and actively maintained a free, open-source project called **Sokrates**<sup>3</sup>. Sokrates

---

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

generates reports with essential source code insights (see Figure 3). Its reports are designed to be user-friendly, making it easy for a broader audience to navigate and understand their source code and its history. Sokrates is designed with an **architect's x-ray vision**, allowing you to **zoom in and out** of source code landscapes. It provides a **high-level overview** of the IT landscape, summarizing data from various teams and groups while also letting you dive deep into the **code-level details**. This dual functionality makes it the perfect sidekick for **CTO-level** strategy powwows and **developer-level** code critiques.

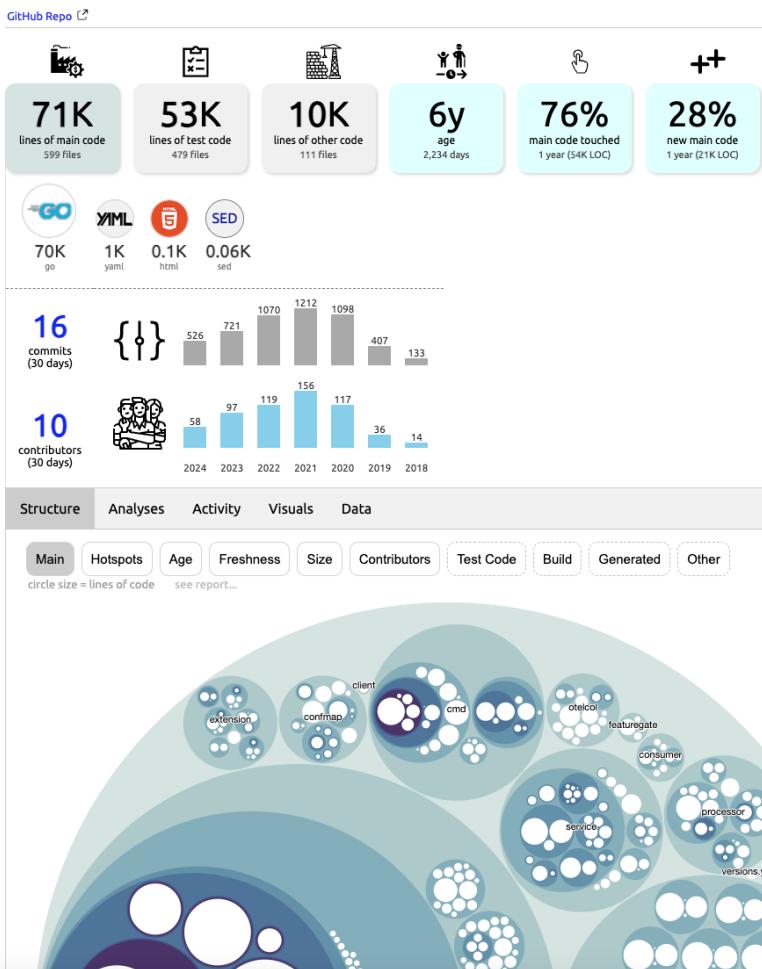


Figure 3: Screenshot of a Sokrates report.

Show me, don't tell me, you said? All right, for a more entertaining look at what Sokrates can do, check out the Sokrates examples. Here are some blockbusters:

- **Apache Software Foundation Repositories<sup>4</sup>:** An epic saga of over 1,000 repositories with more than 180 million lines of code, 22,000

<sup>4</sup>[https://d3axxy9bcypv7.cloudfront.net/asf/\\_sokrates\\_landscape/index.html](https://d3axxy9bcypv7.cloudfront.net/asf/_sokrates_landscape/index.html)

contributors, and 2.4 million commits.

- **Facebook/Meta OSS Repositories<sup>5</sup>:** A thriller with 800 repositories, 120 million lines of code, 20,000 contributors, and over 2 million commits.
- **Microsoft OSS Repositories<sup>6</sup>:** A drama featuring over 2,400 repositories with more than 100 million lines of code, 18,000 contributors, and 1.2 million commits.
- **Google OSS Repositories<sup>7</sup>:** A blockbuster with over 1,600 repositories, more than 200 million lines of code, 27,000 contributors, and 2.4 million commits.
- **Linux Source Code<sup>8</sup>:** A classic with 178 repository sub-folders, more than 23 million lines of code, 17,000 contributors, and 1.7 million commits.
- **Amazon OSS Repositories<sup>9</sup>:** A thriller with over 2,700 repositories, more than 130 million lines of code, 13,000 contributors, and 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:** Perfect for sleuthing how teams build CI/CD pipelines.
- **Dockerfile Scan:** Creates a tech radar of runtime technologies.
- **GitHub API Pull Request Analyses:** To identify deployment frequency.

Feel free to use these or similar tools, but I encourage you to experiment with your source-code analyses as well.

### 5.1.2: Example 2: Public Cloud Usage Analytics

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

---

<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)

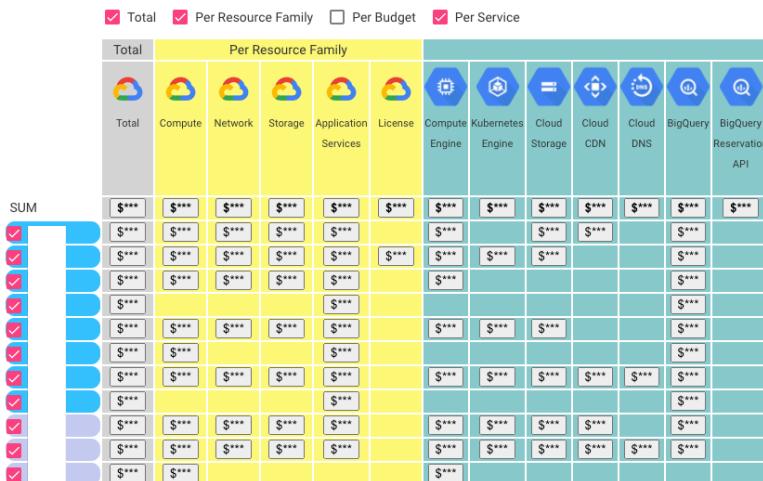
<sup>7</sup>[https://d3axxy9bcycpv7.cloudfront.net/google/\\_sokrates\\_landscape/index.html](https://d3axxy9bcycpv7.cloudfront.net/google/_sokrates_landscape/index.html)

<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)

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 an anonymized 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.*

### 5.1.3: Example 3: Financial and Vibrancy Analytics

Finance departments are like Sherlock Holmes in the business world—super data-driven and always on the case with high-quality data that could be a goldmine for architects. Beyond the usual suspects of costs, budgets, and other dry financial stuff, I've discovered they also track the fun stuff, like vibrancy and usage levels. This data is not just interesting; it's invaluable for architects' work.

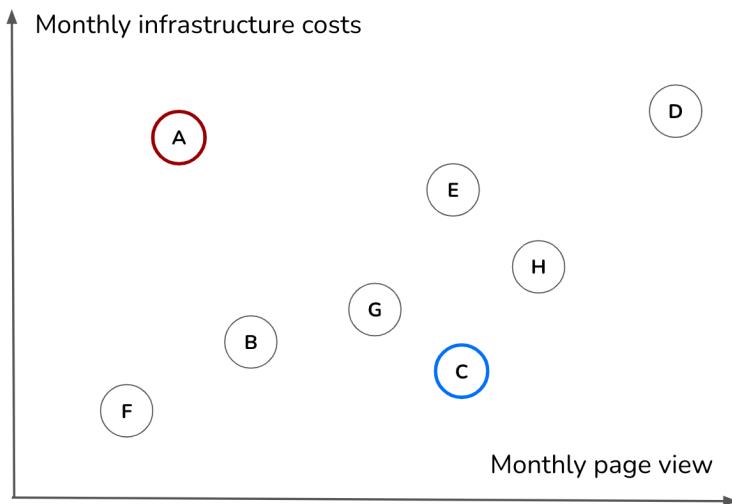
<sup>10</sup><https://aws.amazon.com>

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

<sup>12</sup><https://azure.microsoft.com/>

These finance operatives need this juicy data to, for instance, link the performance of their financial systems with how much they're being used. This kind of usage data is a secret weapon for architecture discussions. For instance, by connecting systems' usage levels and vibrancy with their public cloud costs, we can uncover hidden areas of improvement and inefficiencies (Figure 5).

So, next time you're knee-deep in architectural plans, don't forget to call the finance for top-notch data insights!



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

## 5.2: Requirements For A Lightweight Architectural Analytics

Lightweight Architectural Analytics should be a central place with **authoritative, relevant, and curated data** about the organizational technology landscape. Technically, you can implement Lightweight Architectural Analytics tools like those discussed in the previous section, using simple resources 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. A solid setup will make it easier for more people to access and benefit from the data, turning it into a real asset rather than a digital junk drawer.

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

- **It is the single point of truth** for all relevant architectural data. People should be able to go to one place and get 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 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 informs vital decisions.
- **It is built like a map.** Maps are some of the most crucial documents in human history—they help us store and exchange knowledge about space and place. One thing all maps do is provide readers with a **sense of orientation**. And that, in a nutshell, is what Lightweight Architectural Analytics should offer people in your organization: a sense of orientation in a waste space of technology, organizational, and business topics. The map metaphor is also helpful, as maps come with **multiple layers**. Similarly, the architecture of Lightweight Architectural Analytics should give readers data layers about systems that describe their sizes, connections, quality, security, or human activity. It's like having a trusty map that shows you where the treasure is and warns you about the dragons.

## 5.3: Building Lightweight Architectural Analytics

While each organization has its own quirky set of data, here are some tips I've found helpful in forming the Lightweight Architectural Analytics:

- **Start with the source code.** My motto is "*Talk is expensive. Show me the code.*" Because let's face it, code never lies—people, on the other hand, might forget a detail or two. I scan source code as soon as possible using tools like [Sokrates](#)<sup>13</sup>. Modern IT enterprises store almost everything as code. It's the richest and most up-to-date documentation on what's happening. Quick source code scans can reveal that your "simple" system is actually a digital spaghetti monster.
- **Connect with finance and governance teams.** My second motto is "*Follow the money!*" You'd be amazed at what you can learn from finance data (minus the sensitive parts, like revenue projections—let's keep those secrets safe). Cloud billing reports and tech usage trends are collected anyway. Extract and connect these to get new insights without pestering people for more details.
- **Maintain a culture of transparency.** A culture of transparency means having an organizational environment where open communication, honest information sharing, and accountability are actively encouraged and practiced at all levels. Such a culture allows simplicity, as you need less complex authorization mechanisms.
- **Own the curation.** People need to trust your data like they trust their morning coffee to wake them up. Spend time understanding data sets, curate them, and ensure they're consistently presented. Think of yourself as the master curator and chief UX designer of Lightweight Architectural Analytics.
- **Use simple and easy-to-maintain infrastructures.** For example, I publish the results of Sokrates analyses and other data tools as static resources on our enterprise GitHub pages. Avoid the headache of complex databases and backend software. In the [Architecture Dashboard Examples repository](#)<sup>14</sup>, you'll find the source code for building the Architecture Data Dashboard. The dashboard is a

---

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

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

simple static website generated from JSON files and published via GitHub pages<sup>15</sup>.

These tips might save you from drowning in the sea of data chaos and make your architectural life a bit smoother—or at least give you a few laughs.

---

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

## 5.4: Using Lightweight Architectural Analytics

Lightweight Architectural Analytics can churn out data by the bucketful. Think of it like an atlas or a map—it's great for finding your bearings and understanding the lay of the land. But, with the **right mindset**, you can turn that data into a treasure trove of insights.

Interpreting and using data requires a bit of effort—think of it as a **detective game** where the data holds the answers, but you need to come armed with the right questions. Here are some of the questions you should ask when you've got a pile of data at your fingertips:

- **Are we all rowing in the same direction?** Source code overviews, public cloud usage explorers, or tech radars can highlight when systems and teams are out of sync, sparking heated debates that lead to real action.
- **Are we making the most of our technology?** Comparing usage trends between teams can reveal fascinating outliers—both the virtuosos and those who are... let's say, still tuning their guitars.
- **Are there signs our systems might need a little TLC (tender, loving care)?** Look out for oversized systems, rampant duplication, and files that go on longer than your Aunt Marge's vacation slideshows.
- **Productivity trends: is more really more, or is more actually less?** For instance, comparing the number of git merges to the number of developers can reveal if our dev processes are scalable. When scaling up teams, we aim to speed up delivery, but without proper structure, we might end up with a digital mosh pit.
- **Are we collaborating the way we want to?** Repository analysis can reveal team topologies and unwanted dependencies. Sometimes, teams collaborate like a well-oiled machine; other times, it's more like a group project in high school.
- **Are we working on what we really want to?** We may aspire to innovate, but if we're spending most of our time wrestling with legacy maintenance, we might need to rethink our priorities.

So there you have it. The data's ready to spill its secrets—you need to know the right questions to ask. So, what is your question?

## 5.5: To Probe Further

- Online Appendix Software Tools: Examples and Screenshots<sup>16</sup> screenshots of concrete tools I built as a part of Lightweight Architectural Analytics websites.
- Online Appendix Building Lightweight Architectural Analytics<sup>17</sup> a few practical tips on building lean architecture dashboards and documents using simple, widely available tools.
- Open-source architecture dashboard examples<sup>18</sup>
- Sokrates<sup>19</sup>, an open-source polyglot source code examination tool

---

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

<sup>17</sup><https://grounded-architecture.io/data-website>

<sup>18</sup><https://zeljkoobrenovic.github.io/grounded-architecture-dashboard-examples/>

<sup>19</sup><https://sokrates.dev>

## 5.6: Questions to Consider

Using data can significantly improve the efficiency and impact of an architecture practice. Ask yourself the following questions:

- *What steps would you take to create an Lightweight Architectural Analytics 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: Collaborative Networks



image by mostafa meraji from pixabay

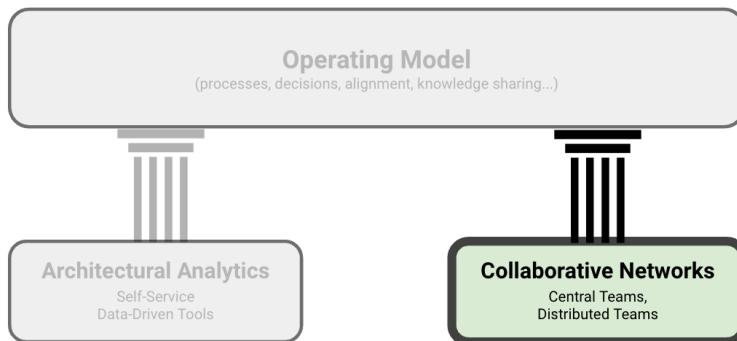
**IN THIS SECTION, YOU WILL:** Understand that an architecture practice is all about people and get tips on creating organizational structures that support a practical IT architecture practice.

**KEY POINTS:**

- Developing an architecture practice requires having competent, empowered, and motivated architects. An 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.

Good architects are a rare breed. They bridge the gaps between business, product, organizational, and technology issues. They're like the Swiss Army knives of the tech world. Hiring architects is like searching for a unicorn who's also a full-stack developer with a knack for diplomacy. They need not only in-depth technical knowledge but also domain-specific and organizational knowledge.

So, you cannot just 3D print your architects or hire them in buckets. But you can carefully organize, empower, and leverage this scarce talent (Figure 1).



**Figure 1:** The Grounded Architecture framework: Collaborative Networks.

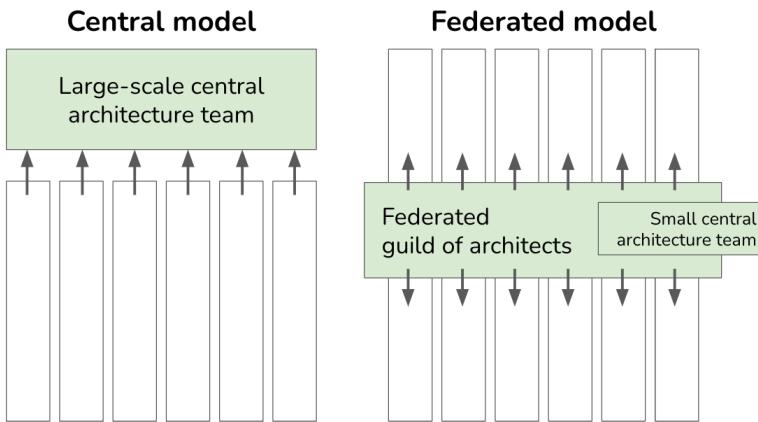
A strong network of people doing architecture is crucial for any real impact. In simpler terms, **Strong Architecture = Strong Architects**.

In my recent escapades, I worked with two teams of architect teams: a

small **central architecture team** and a **cross-organizational distributed virtual team**. The central team is like the wise elders, guiding and supporting the rest of the organization. The distributed virtual team, on the other hand, is a merry band of rebels, working locally but also connecting across the organization, increasing transparency, building networks, and implementing change. Together, such a mix of central and distributed teams can create a robust **collaborative network** driving change across an organization.

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

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



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

The **central model** is like a tightly run coffee shop. A large central team sets the rules, approves new work, and ensures everyone follows the script. Development teams in this model are like customers with no coffee-making skills, relying entirely on the central baristas. The central team handles infrastructure, operations, and security—essentially controlling the caffeine supply chain.

The **federated model** is more like a coffee co-op, where each team has its own barista. A small central team or an **architecture center of excellence (CoE)** might provide high-level guidance, but the real magic happens locally. These baristas (architects) are embedded in development teams, facilitating high-level planning and acting as on-demand service providers.

The **federated model** is a favorite among **cross-functional DevOps cultures**. It integrates infrastructure, operations, and security into its

---

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

brews. Architects in this model focus on **facilitation and enablement**, and stay close to the action.

## 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 Small Central Architectural Team, 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.

Think of it as a symphony orchestra, where the central team is the conductor, and the guilds and virtual teams are the talented musicians playing different instruments. Sure, each musician can play their part solo, but without the conductor, it might sound more like a chaotic jam session at your neighbor's garage.

I prefer the **hybrid team structure** as it scales better 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**.
- Having some **capacity on the central level** serves as a **catalyst**, helping people at local levels to do their jobs 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. However, it's crucial to recognize unique value a central architecture team.



image by simonkr from istock

Such a team, in addition to the doing architecture work, can be instrumental in covering these often overlooked responsibilities:

- **Build and maintain the architecture [Lightweight Architectural Analytics](#).** With all AI advances, it will not build itself! You need clear ownership, curation, and technical support to make it work.
- **Promote data-informed decision-making.** It's not enough to have data; you've got to *use* it. This is like trying to convince your grandpa to use a smartphone—challenging but doable. Architects should be tech-savvy wizards who show everyone else how things are done, making data the critical actor of every decision.
- **Proactively identify, connect, and maintain relationships with all relevant stakeholders.** Think of architects as the social butterflies at a high school reunion—they've got to bridge all the cliques. Connecting different organizational units and stakeholders is their superpower.
- **Build internal architecture communities and guilds.** Organizing rituals and gatherings takes effort.

While guilds and virtual teams can handle many of these activities, their voluntary and sometimes laid-back nature makes their support less reliable. The central architecture team can step in like the dependable backup

generator, taking full long-term ownership and ensuring continuity, even if the community vibe fizzles out.

### 6.2.2: Architecture Guilds & Virtual Architecture Teams

I've always found it crucial to rally passionate troops about architecture through a guild, a community of interest, or a virtual team. After all, who doesn't love a good architecture geek meet-up?



image by sdi productions from istock

Our guilds or virtual team members typically double as architects or tech leads in their respective departments. But they also moonlight by collaborating with their counterparts from other areas. These **peer-to-peer communities** are collectively responsible for spotting and nurturing architectural talent, mentoring, and helping each other out of sticky situations.

If you end up having an overabundance of guilds and teams, you can try splitting them into different cliques:

- **General or core teams:** These folks could tackle various general architectural topics.

- **Specialist teams:** These teams can focus on specific parts of the tech stack, such as native mobile apps, web frontends, or public clouds.
- **Strategic initiatives teams:** These teams work on the big picture, such as data strategy, public cloud strategy, transactions, or verticalization.

Connecting the central and distributed teams is essential. In most places I've worked, we organized gatherings to facilitate regular alignments, such as:

- **Regular (e.g., bi-weekly) forums:** Here, updates are shared, architectural spikes are announced, and decisions are debated.
- **Annual or bi-annual summits:** These are the architecture equivalent of Comic-Con, with several days of intense knowledge sharing and workshops.
- **Ad-hoc workshops:** These focus on specific topics and deep dives into niche subjects.

While the central team can provide essential support, all communities must take the initiative and get many people involved in these events. It's crucial that people are **active participants** rather than passive spectators to ensure more engagement and commitment. So, get ready to roll up your sleeves and dive in, because architecture is a team sport, and everyone needs to play their part!

## 6.3: Building Collaborative Networks

While each organization will need its unique approach, here are some tips I found helpful when forming architecture teams and building Collaborative Networks:

- Before making grandiose plans for reorganizations, **connect with the people already doing architectural work** in an organization. Bring together all the critical tech leaders, regardless of their position and title. Being well-connected to these folks will be crucial in any architecture organization, so this effort is never wasted.
- If creating a virtual team is a part of your architecture strategy, move away from the informal community of practice and start **building a team with more accountability and responsibility**.
- **Connect with non-architecture people** early to gain broader support for building architecture teams and guilds. Being well-connected to these stakeholders is crucial to have strong collaborative networks.
- **Avoid hiring a digital hitman**<sup>2</sup>. Instead, invest in growing internal talent. Think of it this way: architects need to know the technology, the domain, and the organization.
- **Externalize**. Reach out and connect. Participate in external events. Publish your work. Being strong externally can help you grow and attract architectural talent. Everyone wants to join the band when you're rocking the stage.

---

<sup>2</sup><https://architectelevator.com/transformation/dont-hire-hitman/>



image by chantellev from pixabay

## 6.4: To Probe Further

- Agile and Architecture: Friend, not Foe<sup>3</sup>, by Gregor Hohpe, 2020
- Crafting the optimal model for the IT architecture organization<sup>4</sup>, by Christian Lilley et al., 2022
- Developers mentoring other developers: practices I've seen work well<sup>5</sup>, by Gergely Orosz, 2022

---

<sup>3</sup>[https://architectelevator.com/transformation/agile\\_architecture/](https://architectelevator.com/transformation/agile_architecture/)

<sup>4</sup><https://www.mckinsey.com/business-functions/mckinsey-digital/our-insights/tech-forward/crafting-the-optimal-model-for-the-it-architecture-organization>

<sup>5</sup><https://blog.pragmaticengineer.com/developers-mentoring-other-developers/>

## 6.5: Questions to Consider

It is difficult to overestimate the importance of people for an 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:

- *Do you have a strong network of architects across the organization?*
- *Which central, federated, or hybrid model best represents your current an architecture practice? 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 an architecture practice 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: Operating Model: General Principles



image by marish from istock

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

**KEY POINTS:**

- The Operating Model is a system of processes and agreements enabling architects to do everything architecture typically does, leveraging Lightweight Architectural Analytics and Collaborative Networks to create a data-informed, organization-wide impact.
- Examples of activities include: supporting teams in their daily work; tracking tech debt; 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 an architecture practice, I use as a starting point these [two pieces of advice from Gregor Hohpe](#)<sup>1</sup>:

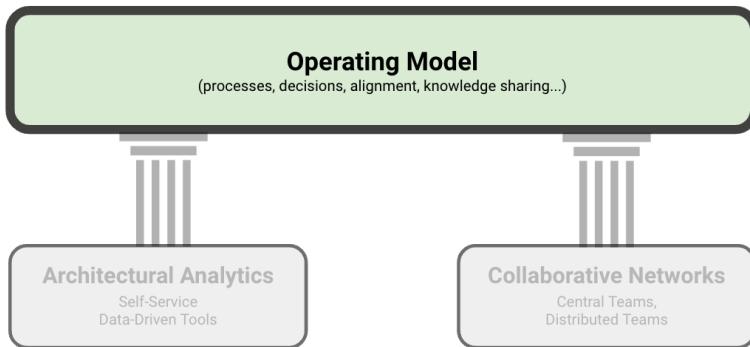
- “*Your architecture team’s job is to solve your biggest problems. The best setup is the one that allows it to accomplish that.*”
- “*Your organization has to earn its way to an effective an architecture practice. You can’t just plug some architects into the current mess and expect it to solve all your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

Considering Gregor Hohpe’s previous two points, I approach defining an architecture practice with the mindset that there is no one-size-fits-all method. 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, it’s crucial to develop **explicit agreements** and “rules of engagement” with key stakeholders. This collaborative approach is essential to create a sustainable and practical an architecture practice.

---

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



*Figure 1: The Grounded Architecture framework: Operating Model.*

This section outlines some lessons I learned when defining IT architecture operating models. The Operating Model (Figure 1) is a part of the Grounded Architecture framework that defines a set of **processes and agreements** that allow architects to do everything an architecture practice typically does. The model should leverage Lightweight Architectural Analytics and Collaborative Networks to develop a data-informed, organization-wide impact. Lightweight Architectural Analytics and Collaborative Networks provide a basis for data-informed decision-making that is well-embedded in the organization.

## 7.1: Examples of Architecture Activities

An Operating Model enables a structured and strategic approach to an architecture practice within the organization.



image by brauns from istock

Here are examples of the activities I have been engaged in with architects to provide a clearer understanding of what I mean by an operating model.

- **Designing Mechanisms for Teams to Make Better Decisions:** These mechanisms involved creating global decision-support frameworks such as advisory forums facilitating informed discussions across teams. For compliance-sensitive projects, we establish formal design authorities. Additionally, we develop team-specific mechanisms, like escalation paths, to resolve decision conflicts effectively (e.g., when teams disagree on a common messaging middleware).
- **Supporting Teams in Their Daily Work:** This support entailed integrating into key team activities and aligning architectural work with team rituals to provide timely support. We assisted teams during all critical phases, such as reviewing architecture proposals before the commencement of a project or sprint, ensuring alignment with overall architectural standards.

- **Supporting Planned New Initiatives and Projects:** Ensuring seamless alignment between projects that require multi-team collaboration is crucial. We worked to facilitate communication and coordination, ensuring all teams are on the same page regarding project goals and requirements.
- **Supporting Teams in Dealing with the Legacy Landscape:** We provided data and insights about the legacy landscape, identifying problematic areas such as frequently changed, low-quality, untested legacy code. We helped define scenarios and roadmaps for legacy modernization, ensuring a structured approach to updating and maintaining legacy systems.
- **Tracking Tech Debt and Defining Tech Debt Reduction Programs:** This involves creating a centrally aligned backlog of technical debt and defining programs for its reduction. We integrate these programs into the planning processes to ensure that tech debt is managed proactively and effectively.
- **Performing SWOT and Other Analyses of Platforms and Systems:** Conducting deep dives to understand specific areas of the technology landscape. We performed SWOT (Strengths, Weaknesses, Opportunities, Threats) analyses and other assessments. These analyses helped in creating comprehensive plans and roadmaps for improvement.
- **Standardizing Processes and Documentation:** We defined standard templates for key documents such as Architectural Decision Records (ADRs), Technical Design Reviews (TDRs), and common diagrams. This standardization ensures consistency and clarity across all architectural documentation.
- **Supporting Merger and Acquisition (M&A) Activities with Expertise and Analyses:** We provided analyses, recommendations, and integration planning for mergers and acquisitions. Such support ensures that architectural considerations are well-integrated into M&A activities, facilitating smoother transitions and integrations.
- **Defining Key Technology Strategies:** We contributed to the development of essential technology strategies, including those for Cloud, Data, and Platforms. These strategies provide a clear roadmap for technological development and investment, ensuring alignment with business goals.
- **Defining Vision and Direction of Technology:** In collaboration

with Engineering Leaders, we created a sustainable organizational setting that aligns with the overarching technology strategies. This work involved setting a clear vision and direction for the technology landscape within the organization.

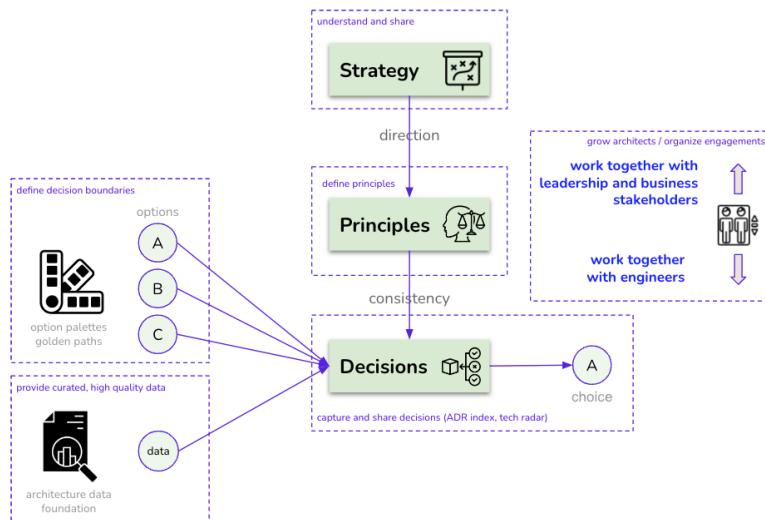
## 7.2: Guiding Principles for Architectural Excellence: Policies, Autonomy, and Engagement

In this section, I address different guiding principles of architectural work:

- Our **operating framework** always emphasizes a **collaborative** and **supportive** approach. Architects should empower development teams to make most decisions while ensuring strategic alignment and minimal compatibility. Architects should engage early in processes to **avoid bureaucratic delays**, focus on constant motion between daily support and strategic tasks, and use data to inform decisions.
- The **distributed decision-making** model promotes team autonomy complemented by high transparency and alignment, guided by principles that balance autonomy with global consistency.
- The “Golden Paths” concept enhances uniformity and efficiency.

### 7.2.1: High-Level Operating Framework

While exact activities and their scope will depend on an organization setting and will change over time, I usually followed a common operational framework in daily work inspired by Gregor Hohpe’s strategy-principles-decisions model (Figure 2).



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

Here are the key characteristics of this operating framework:

#### Engagement mindset:

- Architects engage with stakeholders and teams in a **collaborative and supportive manner**.
- Architects aim to **empower the teams** so that they make most of the decisions.

#### Contributions of architects:

- Bring relevant data to inform decisions leveraging [Lightweight Architectural Analytics](#).
- Define **decision boundaries** to enable minimal compatibility and strategic alignment (e.g., golden paths or tech stack constraints).
- Define **fundamental principles** to facilitate consistency in decision-making.
- Share and generalize lessons learned via [Collaborative Networks](#).

#### Social dynamics 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.

#### Shift left:

- Avoid **formal bureaucratic approval processes**, where architects appear too late and are frequently busy approving trivial decisions.
- 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. Think of it as having the architects as early birds catching the architectural worms, making big changes before the day officially starts.

### 7.2.2: 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, 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.

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

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., golden paths or tech stack constraints).

### 7.2.3: General Architecture Decision Policy

Distributed decision-making scales well, but it can lead to chaos if entirely uncoordinated. Some decision policies are needed. Inspired by the famous [Netflix expense policy](#)<sup>2</sup>, “Act in Netflix’s best interests”, I frequently argued that architecture decision policy could similarly be summarized in six words:

---

<sup>2</sup><https://hbr.org/2014/01/how-netflix-reinvented-hr>

“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 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): 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 by external learning resources (e.g., books, tutorials) significantly helps find and grow experts.
- **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:** If the ownership of components changes (e.g., another team is taking it over), would your choices fit with other components from other areas?
- 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 can resonate well

with many people and can encourage them to be more thoughtful when making decisions.

### 7.2.4: Golden Paths

I have found that the concept of Golden Paths provides an excellent ground to **drive alignment and collaboration** in architecture activities. Golden Paths is an approach utilized to streamline and unify the development process within a software ecosystem, aiming to tackle fragmentation and foster consistency, inspired by [Spotify's implementation<sup>3</sup>](#). Golden Paths can be described as “opinionated and supported” routes developers can follow to build systems efficiently and effectively.

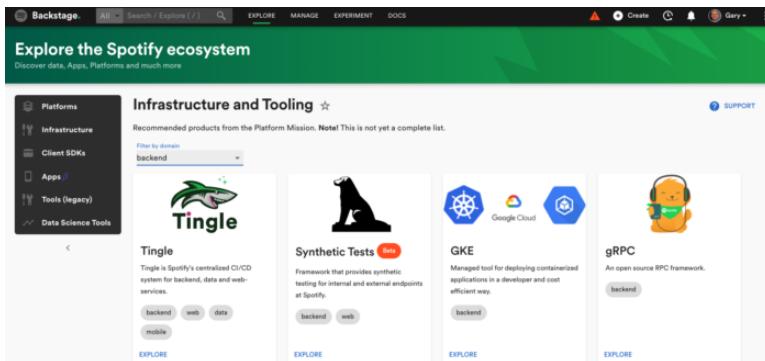


image by engineering atspotify.com

Golden paths provide a solid **foundation for aligning** architecture activities, serving as a common target of work for Guilds and central architectural teams. Rather than being solely knowledge-sharing entities, **guilds** can be empowered to **develop golden paths**, serving as an excellent catalyst for more effective community engagement. This approach not only enhances the role of guilds but also increases the adoption of golden paths as they are created collaboratively.

Golden Paths can be crucial to an organization’s IT development landscape as a deliberate and strategic effort to promote **uniformity, efficiency, and reliability**. By advocating for a set of preferred technologies and practices that are **well-supported, secure**, and aligned with the

---

<sup>3</sup><https://engineering.at spotify.com/2020/08/how-we-use-golden-paths-to-solve-fragmentation-in-our-software-ecosystem/>

organization's broader objectives, Golden Paths can guide developers to build less fragmented, and faster-to-develop software. Ultimately, this leads to higher-quality and more maintainable IT systems.

## 7.3: Embracing Diversity

When building architecture guilds and virtual architecture teams, it's crucial to acknowledge that organizational units have diverse structures and sizes. In big organizations, **embracing diversity** is a prerequisite to having a broad impact.

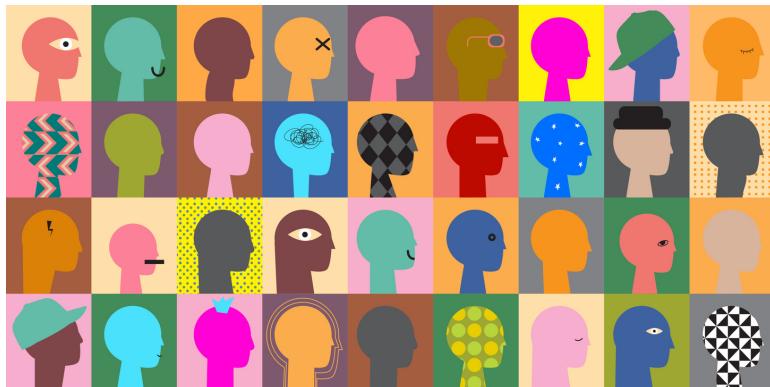


image by annaspoka from istock

There is no one-size-fits-all solution for assigning architecture responsibilities within departments. Based on Gregor Hohpe's view of architects and their teams' relationships, I've generally encountered three types of team-architect systems:

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

Remember, there is no magic bullet. Different structures work for various organizations; sometimes, the best solution is a mix of these approaches.

## 7.4: Setting Boundaries

One of the amusing challenges with setting up an architecture practice 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. We need to **set boundaries** so that we can focus on what’s important rather than becoming frazzled by what’s not.



image by ingenui from istock

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 an architecture practice isn't supposed to do:

1. **Organizational Awareness and Connections:** Teams should know all relevant stakeholders and actively engage with them. This knowledge should include 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, an 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 which is a more

scalable approach. 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 practice goes from a sparkly unicorn to a well-oiled machine!

## 7.5: To Probe Further

- Scaling the Practice of Architecture, Conversationally<sup>4</sup>, by Andrew Harmel-Law, 2021
- Scaling Engineering Teams via RFCs: Writing Things Down<sup>5</sup>, by Gergely Orosz, 2022
- Transformation Agents: An Engagement Model<sup>6</sup>, by Gregor Hohpe, 2022
- Would you like architects with your architecture?<sup>7</sup>, by Gregor Hohpe, 2021

---

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

<sup>5</sup><https://blog.pragmaticengineer.com/scaling-engineering-teams-via-writing-things-down-rfc/>

<sup>6</sup><https://architectelevator.com/transformation/transformation-engagement-model/>

<sup>7</sup><https://architectelevator.com/architecture/organizing-architecture/>

## 7.6: Questions to Consider

Your an 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 Operating Model 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?*
- *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?*
- *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: Cooperation-Based Operating Model: Six Simple Rules



image by nanostockk from istock

**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 offered fresh air for my vision of an 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**.

The authors developed the Six Simple Rules approach as a practical solution for today's complex business environment. The rules in the book are based on the premise that the key to managing complexity is the **combination of autonomy and cooperation**. The book advocates for the setup of organizational structures that harmonize, empowering individuals with more autonomy to act. This approach is about trusting the capabilities of the organization's people to handle complex problems, fostering a cooperative and efficient work environment.

In this chapter, I explore how Grounded Architecture and Six Simple Rules are best friends. The Six Simple Rules ideas have been a significant source of inspiration for my work. [Conway's Law](#)<sup>2</sup> shows that the link between organizational structures and IT architecture is like peanut butter and jelly—strong and better together. The Six Simple Rules approach and architectural work are all about managing complexity without getting tangled up.

---

<sup>1</sup><https://www.bcg.com/capabilities/organization-strategy/smart-simplicity>

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

## 8.1: Background: Hard and Soft Management

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.

### 8.1.1: Hard Approach

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.



image by cyano66 from istock

In my experience, IT architecture support in organizations following a hard management approach has the following characteristics:

- **Heavy Reliance on Tools:** The architecture would focus on tools, automation, and workflows that **enforce compliance**, standardization, and control. There might be a strong reliance on enterprise resource planning (ERP) systems, centralized data warehouses, and other structured tools that enable **strict adherence** to predefined processes.
- **Security and Compliance:** There would be a significant emphasis on **compliance frameworks**, security protocols, and risk management systems. These systems would be designed to ensure that all actions are traceable, compliant with regulations, and aligned with the company's **predefined rules and processes**.
- **Bureaucratic Systems:** The architecture might include multiple layers of control and coordination units, such as a risk management system, compliance databases, and innovation management software. Each of these units would be designed to enforce specific aspects of the organization's performance objectives.

### 8.1.2: Soft Approach

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).**



image by alessandro biascioli from istock

In my experience, IT architecture support in organizations following a soft management approach has the following characteristics:

- **Employee Empowerment:** Systems would likely be designed to empower employees by providing the tools they need to succeed independently, such as access to real-time data, self-service analytics, and systems that encourage innovation and creativity.
- **Employee-Centric:** The architecture would focus on tools facilitating communication and knowledge sharing. These tools might include collaboration platforms, social intranets, and user-friendly interfaces that enable employees to interact more effectively.
- **Personalization:** Systems might be more personalized, offering customization options that align with individual preferences and needs. This personalization could include customizable dashboards and personalized workspaces.

## 8.2: Collaboration Approach

Hard and soft management 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.

The Six Simple Rules approach emphasizes that in today's business environment, you must set up **organizational structures based on cooperation**. More specifically, the Six Simple Rules approach involves the interplay of **autonomy** and **cooperation**. The authors emphasize the critical 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.

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.

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.

## 8.3: 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 scyther5 from istock

### 8.3.1: Guidelines for Understanding Performance

To genuinely understand performance, consider the following principles:

- Trace performance back to behaviors, understanding how these behaviors influence and combine to produce overall results.
- Utilize observation, mapping, measurement, and discussion to gain insights.
- Comprehend the context of goals, resources, and constraints within which current behaviors are rational strategies for people.
- Discover how your organization's elements—structure, scorecards, systems, and incentives—shape these goals, resources, and constraints.

### 8.3.2: Leveraging Architecture Practice

Architecture practice can significantly aid in understanding organizational behaviors through:

- Establishing a **Lightweight Architectural Analytics** with an overview of various data sources to reveal where activities occur, visible trends, and cooperation among people.
- Utilizing the **Collaborative Networks** to connect individuals and enable them to learn about activities in different parts of the organization.

## 8.4: 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

### 8.4.1: Guidelines for Reinforcing Integrators

To strengthen integrators within your organization, consider these strategies:

- **Use emotions** to identify candidates. Feelings provide crucial clues for analysis and can act as symptoms of integration issues.
- **Identify operational units** that can be integrators among peer units due to their specific interests or power.
- **Remove managerial layers** that do not add value and reinforce others as integrators by eliminating specific rules and relying

on observation and judgment over metrics when cooperation is involved.

### **8.4.2: Enhancing Integrators Through Architecture Practice**

Architecture practice can play a vital role in reinforcing integrators by:

- Utilizing the **Collaborative Networks** to help identify and connect integrators, leveraging their work effectively.
- Emphasizing architects as critical integrators and integrator role models within the organization, defining them as essential components of the organizational “superglue.”
- Establishing a **Lightweight Architectural Analytics** to support integrators with data and insights, enabling them to perform more informed and effective work.

## 8.5: 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 prostock\_studio from istock

### 8.5.1: Guidelines for Increasing Power Quantity

To enhance the quantity of power within your organization, consider these actions recommended by the Six Simple Rules approach:

- When making design decisions that could shift the balance between central and unit power, functions, and line managers, ensure that some parts of the **organization benefit from new power bases**. This approach helps meet complexity requirements and avoids future disruptions from pendulum swings.

- When creating new functions, ensure they have the power to fulfill their roles **without diminishing the power others need** to fulfill theirs.
- When introducing new tools for managers, such as planning or evaluation systems, evaluate whether these tools **act as resources or constraints**. Implementing a few tools creates a critical mass of power, which is more effective than introducing many tools sequentially.
- Regularly **enrich power bases** to maintain agility, flexibility, and adaptability.

### **8.5.2: Enhancing Power Quantity Through Architecture Practice**

Architecture practice can support the increase in power quantity through an operating model that promotes distributed decision-making:

- Through the **Collaborative Networks**, you can enhance decision-making power by distributing architectural decision-making across the organization and embedding it within development teams, which typically have the best insights and most relevant information.
- Additionally, **Lightweight Architectural Analytics**, accessible to all interested members of the organization, can provide data and insights that empower individuals in their daily work.

## 8.6: 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 natnan srisawan from istock

### 8.6.1: Guidelines for Enhancing Reciprocity

Reciprocity involves recognizing that people or units in an organization have a mutual interest in cooperation and that the success of one depends on the success of others. To foster this reciprocity, follow these guidelines:

- **Eliminate monopolies** to ensure no single entity has exclusive control.
- **Reduce resources** to encourage more efficient and collaborative use.
- **Create new networks** of interaction to facilitate better communication and cooperation.

## 8.6.2: Enhancing Reciprocity Through Architecture Practice

Architecture practice can significantly increase reciprocity within an organization:

- The **Collaborative Networks** supports creating new networks of interactions, directly reinforcing reciprocity.
- A **hybrid operating model** relies on the mutual success of an architecture practice and development teams. Architects' **impact** is essential, and their support depends on the feedback from the groups they assist. Integrating this feedback into architects' performance evaluations is crucial for enhancing reciprocity between architecture and other units.

## 8.7: 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**.



image by joe from pixabay

### 8.7.1: Guidelines for Extending the Shadow of the Future

The Six Simple Rules approach recommends four strategies to extend the shadow of the future:

- **Tighten the feedback loop** by increasing the frequency of moments when people experience the consequences of the fit between their contributions.
- **Bring the endpoint forward** by shortening the duration of projects.

- Tie **futures together** so that successful moves are conditioned on contributing to the success of others.
- Ensure people **walk in the shoes they make** for others.

### **8.7.2: Extending the Shadow of the Future Through Architecture Practice**

Architecture practice can play a crucial role in extending the shadow of the future through various methods:

- **Lightweight Architectural Analytics** can create transparency and provide the data necessary to model the future. This data can be used to develop simulations and roadmap options.
- Applying **economic modeling** to architecture decision-making helps describe the future consequences of today's actions, directly supporting long-term planning and decision-making.

## 8.8: 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 to reward those who cooperate.



image by stocksnap from pixabay

### 8.8.1: Guidelines for Rewarding Cooperation

To effectively reward those who cooperate, managers should:

- Go beyond technical criteria and avoid placing blame solely on where the root cause originated. Accept that execution problems arise from various reasons. The smart approach is to reduce rewards for those who fail to cooperate in solving a problem, even if the problem isn't in their direct area, and increase rewards for units that cooperate beneficially.
- Avoid blaming failure and instead focus on blaming the inability to help or seek help.

- Use simple questions to shift managerial conversations, making transparency and ambitious targets resources rather than constraints. This approach helps managers act as integrators, leveraging cooperation and rich information to achieve superior results.

### **8.8.2: Rewarding Cooperation Through Architecture Practice**

Architecture practice can facilitate rewarding cooperation by making it easier for individuals to help others and ask for help:

- A strong **Collaborative Networks** can provide the context and networks necessary for easier collaboration.
- Adding diverse data sources to **Lightweight Architectural Analytics** can create transparency about cooperation opportunities and challenges, supporting a more collaborative environment.

## 8.9: To Probe Further

- Six Simple Rules: How to Manage Complexity without Getting Complicated<sup>3</sup>

---

<sup>3</sup><https://www.bcg.com/capabilities/organization-strategy/smart-simplicity>

## 8.10: 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 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?

# 9: Operating Model: Nudge, Taxation, Mandates

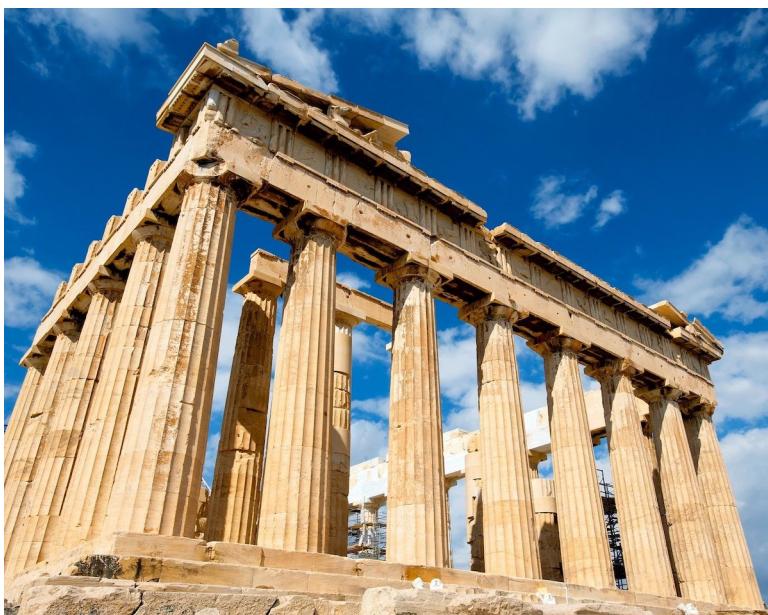


image by nonbirinonko from pixabay

**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. IT architecture governance involves defining the technological infrastructure, setting standards for technology use, and ensuring that all technology-related activities align with the organization's overall goals. This form of governance provides a comprehensive approach to managing technology, ensuring that it supports the organization's strategic direction and operational needs.

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-size-fits-all form of governance, as each organization has unique needs, goals, and environments. Effective governance requires adaptability, collaboration, and a commitment to addressing immediate and long-term issues. It involves balancing various stakeholder interests, anticipating and responding to external changes, and continually refining governance practices to meet evolving demands. This complexity demands a strategic approach, robust communication, and a willingness to innovate and adapt to ensure that governance frameworks remain relevant and effective in achieving organizational objectives.

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.**

## 9.1: 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 liudmila chernetska from istock

A nudge can take many forms, such as a slight change in the environment, a gentle reminder, 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. 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. Choice architecture

refers to how options are presented to individuals, which can significantly influence their choices. It 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 **Lightweight Architectural Analytics** dashboard 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 **Lightweight Architectural Analytics** tools to **highlight areas and issues** we wanted (nudged) people to improve. The **Collaborative Networks** can create mechanisms for sharing experiences, promoting **positive examples**, and capturing lessons learned to help people make better, more informed decisions. In the **Operating Model**, 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.

---

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

## 9.2: Taxation (Economic Incentives)

Governing with taxes is a strategic approach where individuals or departments are not prohibited from making choices or decisions. Instead, they are required to “pay” some form of tax on the resources they use. This system creates a feedback loop that encourages responsible resource consumption and helps optimize overall system efficiency. One practical application of this approach is in managing the costs of public cloud usage within an organization. By cross-charging these costs across various departments or projects, each unit receives a clear signal about their resource consumption. This not only helps in allocating costs accurately but also promotes awareness and encourages efforts to minimize unnecessary usage, effectively preventing the “tragedy of the commons,” where unrestricted access to shared resources can lead to overconsumption and depletion.



image by steve buissinne from pixabay

Compared to nudging, which influences behavior subtly by providing information or setting default choices without imposing direct consequences, taxes introduce tangible consequences. For example, projects that exceed budgeted IT costs due to excessive resource consumption might be canceled. The role of an architecture practice in this form of governance is crucial. It involves ensuring that taxation policies are based on accurate and comprehensive data, using public cloud cost reports and

other relevant information to inform tax rates and policies. Transparency is essential, allowing all stakeholders to understand how and why taxes are levied and providing clear insights and reports detailing the basis of taxation and its impact on resource consumption.

Developing efficient feedback loops is another critical aspect, providing timely and actionable feedback on critical metrics related to taxes and continually refining and optimizing the taxation system. [Lightweight Architectural Analytics](#) plays a key role in this approach by aggregating and analyzing all data related to resource consumption and taxation, generating insights from public cloud cost reports, and guiding decision-making. Collaborative Networks ensures alignment of organizational processes, goals, and working methods with the taxation syst~~2022-06-03-skills.markdownem~~, fostering a culture of responsible resource usage and continuous improvement.

In conclusion, governing with taxes is a robust approach to resource management that balances autonomy with accountability. By implementing a data-driven and transparent taxation system, organizations can optimize resource usage, prevent overconsumption, and drive meaningful change. An architecture practice, supported by strong [Lightweight Architectural Analytics](#) and [Collaborative Networks](#), is essential in achieving these goals and ensuring the sustainability of shared resources.

## 9.3: Mandates and Bans

Governing with mandates and bans involves guiding people by explicitly defining what they should or should not do. In various workplaces, such mandates and bans have played a limited yet important role in defining the broader strategic boundaries of choices available to people. For instance, restricting the use of public cloud providers to specific vendors or adhering to strict privacy and security procedures needs to be explicitly defined and controlled.

Using bans with care and as a last resort is essential to avoid unnecessary blocking or slowing down development and innovation. However, mandates and bans can help clarify critical topics where nudging or taxation would not be sufficient. For example, having clear rules and control mechanisms to avoid breaking privacy or financial laws can prevent unnecessary incidents and damage. Explicitly defined mandates and bans can ensure compliance with important regulations and safeguard the organization's integrity and reputation.



image by tumisu from pixabay

The role of architecture in this form of governance should be to act as a stakeholder but not the sole owner in defining mandates and bans.

These mandates and bans should often be determined collaboratively with other functions, such as security and legal departments. The architecture practice can contribute by **creating clarity and providing transparency**.

**Lightweight Architectural Analytics** is crucial for creating **clarity and transparency**. For example, it can provide insights through security reports or maps of areas in the source code or infrastructure that need monitoring and controlling based on privacy or security requirements. This foundation helps in identifying and mitigating risks, ensuring that the organization's technology landscape aligns with its governance frameworks.

The **Collaborative Networks** can help propagate the decision and ensure its **positive impact and acceptance**. Mandates and bans should not be issued routinely without sufficient explanation. It is crucial to spend time with all stakeholders to explain the **reasons and motivations** behind introducing certain limitations. A strong Collaborative Networks fosters strong connections with key stakeholders, leveraging these relationships to implement changes smoothly. This foundation ensures that governance measures are understood, accepted, and integrated into the organization's culture and practices, enhancing overall effectiveness.

In summary, governing with mandates and bans involves setting clear, explicit guidelines for acceptable behavior and practices. It requires a balanced approach to ensure it does not stifle innovation while maintaining necessary controls. Effective implementation relies on collaboration, transparency, and communication to ensure that all stakeholders understand and accept the governance measures.

## 9.4: 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 to create clarity and provide 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?*

# 10: Transforming Organizations with Grounded Architecture



image by gremlin from istock

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

**KEY POINTS:**

- When a Grounded Architecture framework is in place, it can positively transform 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 framework is in place, it can positively transform an organization's functioning. These categories of impact, aligned with [defined goals](#), are:

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

## 10.1: Executing at Scale

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

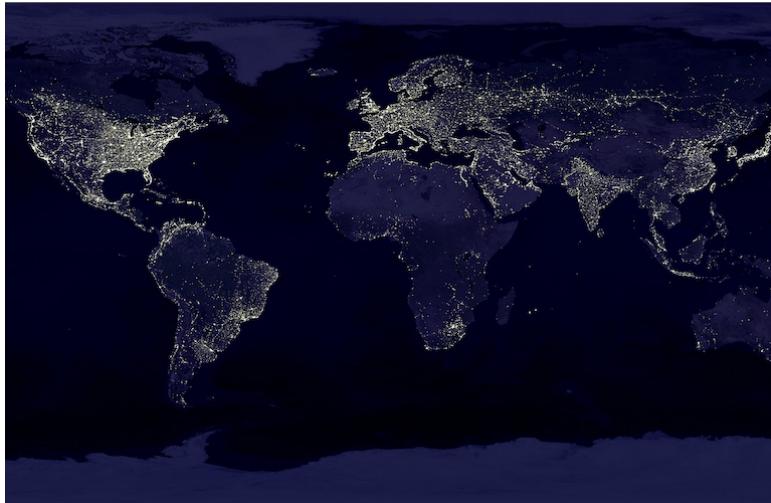


image by wikiimages from pixabay

Each element of Grounded Architecture enables an architecture practice to operate at scale in several ways.

### 10.1.1: Lightweight Architectural Analytics

The Lightweight Architectural Analytics can support large-scale operations by:

- **Self-Service Data Access:** Providing data as a self-service resource allows teams to access the information they need independently, reducing the reliance on manual sharing methods. This service can be facilitated through internal websites or portals where data tools and insights are readily available.

- **Elimination of Meetings:** As architects frequently own some unique pieces of information, in particular related to the big picture and overall dependencies, lots of architecture support goes on providing that information to the teams. Typically that means having meetings, workshops and exchanging lots of messages. With Lightweight Architectural Analytics, we significantly reduced the need for such information-sharing or data-gathering meetings (every time we added a new data app, I got a few hours back in my calendar and exchanged fewer messages). Sharing data via Architectural Analytics dashboards also generates a useful feedback about how data are used, and helps ensuring that data is up-to-date and accessible.
- **Automation:** Automation minimizes manual effort, which is crucial as manual processes do not scale efficiently. Automating data management processes ensures that data can be collected, processed, and analyzed without extensive human intervention.

### 10.1.2: Collaborative Networks

The Collaborative Networks enhances execution at scale by focusing on:

- **Developing Connections:** Building strong relationships at all levels of the organization is crucial. This network facilitates quicker alignment of objectives, efficient information sharing, and swift execution of shared decisions.
- **Speeding Up Alignment:** Effective communication channels and collaborative tools help align teams rapidly, ensuring everyone is on the same page and working towards common goals.
- **Facilitating Shared Decisions:** Enabling a culture where shared decisions are made quickly can enhance the responsiveness and adaptability of the organization.

### 10.1.3: Operating Model

The Operating Model promotes execution at scale by:

- **Distributed Decision-Making:** Distributing decision-making across an organization prevents bottlenecks associated with centralized decision-making processes. This model empowers more people to take ownership and make decisions within their scope, leading to faster and more effective outcomes.
- **Promoting a Collaborative Operating Model:** An operating model that supports distributed decision-making and collaboration ensures that the organization can handle more projects simultaneously without overburdening any single entity or individual.

By leveraging these foundations, we managed the complexities and diversity of numerous projects and teams more efficiently, ensuring scalability and effective execution.

## 10.2: Adaptivity

The second goal, ensuring that an architecture practice can adapt quickly to stay relevant in new contexts, is crucial for maintaining an organization's agility and resilience.

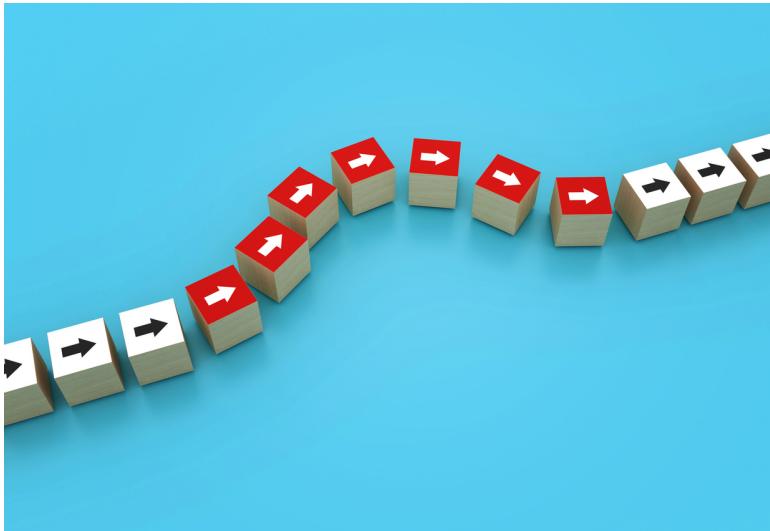


image by porcorex from istock

The Grounded Architecture framework has core elements that provide a highly flexible and adaptive setting. Here are the key drivers of flexibility within this structure:

### 10.2.1: Lightweight Architectural Analytics

The **Lightweight Architectural Analytics** can support adaptability by:

- **Automation and Extensibility:** We used lean and flexible tools that we could swiftly extend and reconfigure to accommodate changes. For example, following acquisitions or mergers, with our lean tools and automation scripts, we were able to integrate all new source code repositories and many other data sources in a matter of hours or days. Automation ensures that data remains relevant and

up-to-date, providing essential connections and feedback tailored to the organization's evolving needs.

- **Configurable Adaptation to Organizational Changes:** We created robust Lightweight Architectural Analytics to adapt its views to the changing realities of different parts of the organization, ensuring that data insights are always aligned with current operational contexts. For instance, we developed tools that enable explorative, on-the-fly aggregation of diverse data sources (e.g., created aggregated source code statistic reports based on repository tags or name conventions).

### 10.2.2: Collaborative Networks

The Collaborative Networks can support adaptability by:

- **Capacity Redistribution:** Collaborative Networks, through a central team, can alleviate the temporary capacity shortages experienced by distributed teams. This flexibility ensures that an architecture practice continue to operate smoothly even during periods of high demand or limited resources.
- **Decentralized Support:** By maintaining a network of well-connected architects, an architecture practice can support the organization effectively without relying solely on a central team. This decentralization fosters resilience and adaptability, enabling architectural guidance and oversight across various teams and projects.

### 10.2.3: Operating Model

The Operating Model can support adaptability by:

- **Flexible Decision-Making:** The Operating Model supports adaptability by promoting a flexible setting and distributing decision-making authority throughout the company. This distribution helps prevent an architecture practice from becoming a bottleneck or a single point of failure.

- **Delegation of Architectural Decisions:** Senior architects can delegate most architectural decisions to teams, allowing them to focus on critical strategic initiatives, such as defining cloud, data, or platform strategies and supporting significant decisions related to mergers and acquisitions. By grounding the architecture with data and people connections, organizations empower senior architects to concentrate on high-impact areas.

This structure enhances an architecture practice's flexibility and adaptivity. It ensured we could respond promptly and effectively to new challenges and opportunities.

## 10.3: 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 cofotoisme from istock

To make the decision process more data-informed and less dependent on opinions, we need to focus on the interplay between three foundational elements. These elements collectively create a robust framework for data-driven decision-making in architectural discussions.

### 10.3.1: Lightweight Architectural Analytics

The **Lightweight Architectural Analytics** ensures the availability and readiness of data needed for decisions, fueling data-informed discussions:

- **Data Collection:** Systematically gather high-quality data on relevant internal and external technology developments.
- **Data Management:** Maintain a well-organized and easily accessible database that supports quickly retrieving relevant data.

- **Data Analytics:** Employ analytics tools to process and interpret the data, providing meaningful insights to support decisions.
- **Real-Time Data Availability:** Ensure that data is updated regularly and available in real-time to inform ongoing discussions and decisions.

### 10.3.2: Collaborative Networks

Data is not enough; you need to make this data available to decision-makers. The **Collaborative Networks** ensures people are well-connected to share information and make decisions via:

- **Expert Network:** Establish and maintain a network of experts and stakeholders who can provide insights on various architectural aspects.
- **Collaboration Tools:** Implement collaboration tools that facilitate seamless communication and information sharing among team members.
- **Training and Development:** Provide training programs to enhance the data literacy and analytical skills of architects and decision-makers.
- **Engagement Practices:** Develop practices for regularly engaging stakeholders and ensuring their input is considered in decision-making.

### 10.3.3: Operating Model

The **Operating Model** provides processes that enable architects to move from opinion-based decisions to data-driven economic risk modeling via:

- **Process Implementation:** Implement processes that guide architects in dismantling hype and buzzwords, presenting problems clearly, and making data-driven decisions.
- **Data Integration:** Develop methodologies for integrating relevant data into discussions, ensuring that data supports and guides the conversation.

- **Economic Risk Modeling:** Create models that translate drivers and data into economic risk assessments, helping to identify the best solutions for the given business context.
- **Decision Support Tools:** Deploy tools and technologies that assist in visualizing data, modeling risks, and evaluating options.

Focusing on these foundational elements and their key actions can transform our architectural discussions into a more data-informed process. This focus ultimately led us to better, more objective decision-making that aligns with our business context and goals.

## 10.4: Maximizing Organizational Alignment

The fourth goal emphasizes that an architecture practice should be a cohesive factor in minimizing misalignments within large organizations.



image by 36clicks from istock

Misalignments often occur in such settings due to various factors, including complex structures and diverse objectives. However, a well-grounded architecture can address and reduce these misalignments through its foundations.

### 10.4.1: Lightweight Architectural Analytics

The **Lightweight Architectural Analytics** improves organization alignment by **creating transparency**. By establishing a robust Lightweight Architectural Analytics, organizations can enhance transparency, which is necessary for building trust and facilitating alignment. This involves making data easily accessible and understandable across different departments, ensuring everyone has the same information and can make informed decisions. Transparent data practices help align objectives and actions, reducing the chances of misalignment.

### 10.4.2: Collaborative Networks

The **Collaborative Networks** improves organization alignment by **facilitating collaboration**. Collaborative Networks focuses on developing global structures that connect employees across various functions and geographies. By making it easier for people to collaborate, share knowledge, and work together, this foundation helps create a unified approach to organizational goals. Collaborative environments foster alignment by ensuring that everyone is working towards common objectives.

### 10.4.3: Operating Model

The **Operating Model** improves organization alignment by:

- Facilitating **pre-decision alignment** via **collaborative decision-making**. This platform plays a pivotal role in minimizing misalignments by enabling individuals and teams working on similar projects or topics to identify each other and collaborate. This early alignment minimizes duplication of efforts and optimizes resource utilization, ensuring that efforts are aligned from the outset.
- Facilitating **post-decision dissemination** via **knowledge sharing and more awareness**. After decisions are made, we can leverage the collaborative networks to ensure that these decisions are communicated across the organization. This widespread dissemination helps all parts of the organization benefit from the insights and lessons learned from one unit, leading to more cohesive and aligned operations.

An architecture practice can reduce misalignments and drive a more unified, efficient, and aligned organization by integrating data transparency, enhanced collaboration, and structured processes for decision-making and dissemination.

## 10.5: Maximizing Organizational Learning

*“Good judgment comes from experience, and experience comes from bad judgment.”* –Fred Brooks

Our 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 and create pragmatic technology recommendations for concrete platforms across the organization based on their understanding.



image by rawpixel from istock

The Grounded Architecture framework offers comprehensive support through its Lightweight Architectural Analytics, Collaborative Networks, and Operating Model to help organizations learn quickly, stay up-to-date with emerging technologies, and recommend technology upgrades.

### 10.5.1: Lightweight Architectural Analytics

Lightweight Architectural Analytics plays a pivotal role in accelerating the learning and adoption of new technologies. Here's how:

- **Facilitating Exploration and Reflection:** Lightweight Architectural Analytics enables individuals and teams to explore new tools and technologies effectively by providing relevant and comprehensive data. Access to this data allows for thorough experimentation, analysis of outcomes, and reflection, which is crucial for understanding and refining new technology implementations. For instance, it may not be easy to calculate the cost of a new cloud service. However, running proofs-of-concept and analyzing collected data can provide good insights.
- **Supporting Informed Decision-Making:** Architects and teams can make informed decisions regarding technology adoption and upgrades with up-to-date data. This minimizes the risk associated with implementing new technologies and ensures that the choices are based on solid evidence and insights.

### 10.5.2: Collaborative Networks

Collaborative Networks enhance learning by creating and maintaining a culture of knowledge sharing. It achieves this through:

- **Spaces for Sharing Knowledge:** Regular update calls, knowledge-sharing sessions, and conferences need to be organized to facilitate the exchange of architectural and technological knowledge. These events are critical for informing the organization about the latest developments and best practices.
- **Maximizing Personal Learning:** Collaborative Networks ensure that individual lessons are transformed into shared guidelines by deriving generalized insights from cross-group cases. This collective intelligence benefits the entire organization and fosters continuous personal and professional growth among team members.

### 10.5.3: Operating Model

The Operating Model integrates learning into daily workflows through structured processes:

- **Embedding Learning into Processes:** Learning is accelerated by embedding it into the organization's processes. This is achieved by defining and distributing knowledge-sharing and lesson-learned processes across the organization. By doing so, learning becomes a seamless part of daily activities rather than an additional task.
- **Systematic Knowledge Capture and Application:** The platform ensures that knowledge is systematically captured, shared, and applied. This approach minimizes overhead while maximizing learning opportunities, enabling the organization to quickly adapt and apply new insights and technologies.

By leveraging Lightweight Architectural Analytics, Collaborative Networks, and Operating Model, the Grounded Architecture framework ensures that learning is continuous, efficient, and embedded within the organization's fabric. This comprehensive support system helps organizations stay up-to-date with emerging technologies and industry trends and fosters a proactive learning environment that drives innovation and growth.

## 10.6: 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 an architecture practice at scale? How valuable could principles of Grounded Architecture be in enhancing 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 an architecture practice? Could the Lightweight Architectural Analytics and Collaborative Networks 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 an architecture practice accelerate this process?*
- *Consider the adaptivity of your organization's an architecture practice. How could your an architecture practice improve it?*
- *Reflecting on the value of the "Lightweight Architectural Analytics" 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 an architecture practice in your organization in the absence of a strong central team? Could implementing a Lightweight Architectural Analytics and well-connected architects help mitigate this?*

# **Part II: On Being Architect**

# 11: On Being Architect: Introduction

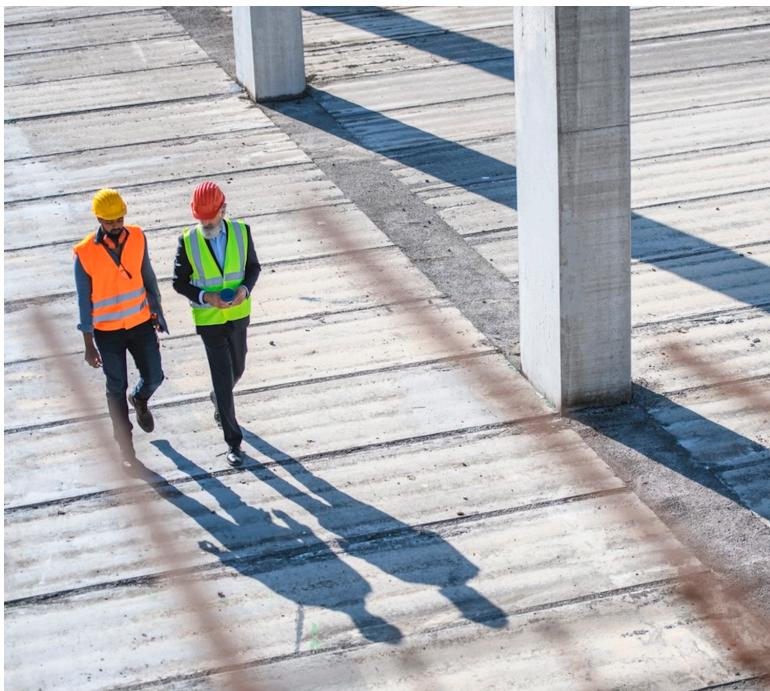
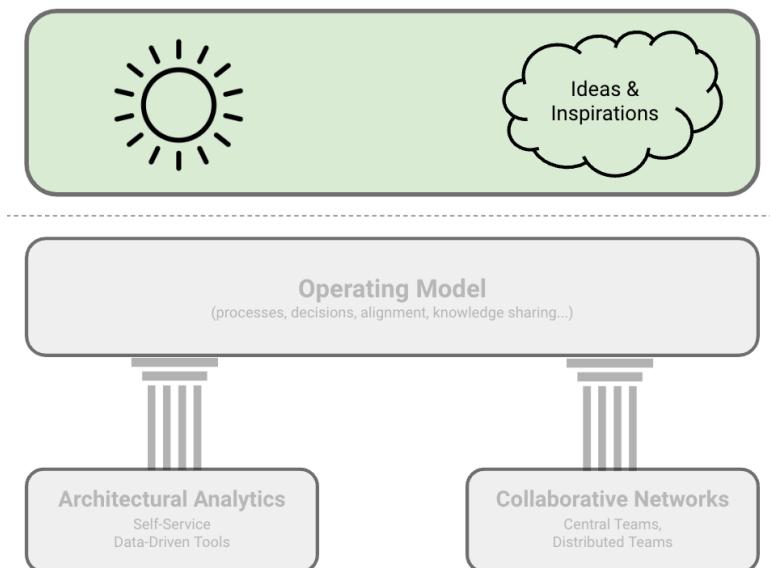


image by azmanl from istock

**IN THIS SECTION, YOU WILL:** Get an overview of lessons I learned on what it means to be an architect in practice.

With this section, we start the second part of our book, which unveils a wealth of practical ideas and inspiration to help you bring the Grounded Architecture framework we outlined in the first part to life. This part of the book will equip you with **valuable tips and insights** for running an IT architecture practice (Figure 1). In complex organizations, you cannot just copy and paste other companies' approaches. Instead of having a rigid structured framework, having a toolbox of inspirational resources is more practical for implementing a successful architectural practice.



*Figure 1: Grounded Architecture Overview: Ideas and Inspirations.*

We begin this second part with several resources about what it means to be an architect. The role of an architect in the IT industry is like a **Swiss Army knife**—multifaceted and indispensable. It requires blending technical expertise, strategic thinking, and interpersonal skills to drive successful outcomes in complex organizational environments. In this part of the book, I offer a deep dive into various perspectives on my view on what it means to be an architect in practice, providing a holistic understanding of the responsibilities and expectations associated with this role.

By examining these perspectives, I hope you will uncover some of the architects' essential qualities and responsibilities. My exploration high-

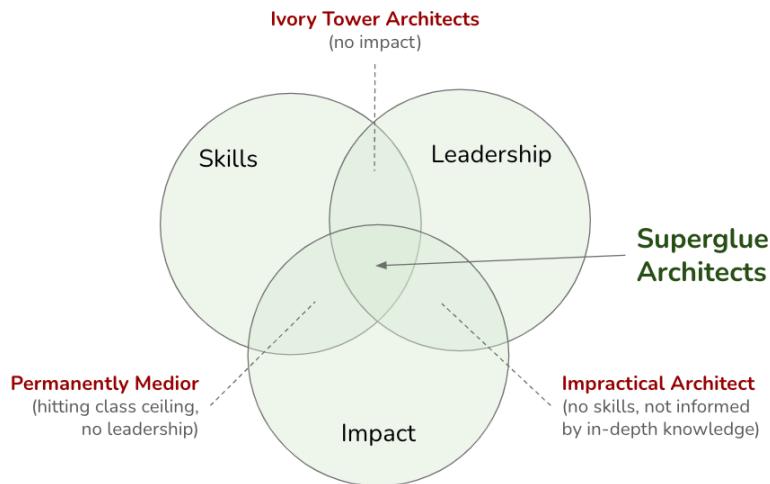
lights the importance of a **well-rounded skill** set and a strategic approach, showing that being an architect is more than just wearing stylish glasses and nodding thoughtfully in meetings. Whether you're an aspiring architect or a seasoned pro, I hope these insights can provide valuable guidance on navigating and excelling in this dynamic and evolving field.

## 11.1: Growing As An Architect

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:

- Skills
- Impact
- Leadership

Architects must have a **minimal “length”** of all of these “legs” to be successful (Figure 2). 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 2:** Architects must have a minimal “length” of all “legs” to be successful.

### 11.1.1: Skills

Architects must have a solid skill set, possessing both knowledge and the ability to apply relevant knowledge in practice. These skills should include technical (e.g., cloud architecture or Kubernetes technology) as well as 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:** like the ability to calm a panicking developer or decode the cryptic language of a business executive,
- **Product development knowledge:** knowing what makes a product tick,
- **Business domain knowledge:** understanding those endless spreadsheets,
- **Decision-making skills:** choosing the right path, even when all options look doomed.

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

### 11.1.2: Impact

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.

### 11.1.3: 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**.
- 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 organization.

I also support Gregor Hohpe's view that **mentoring other architects** is one of the most crucial aspects 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.

## 11.2: Thinking Like an Architect

In IT organizations, architects are the **essential connectors**, often referred to as “super glue.” Their primary function is integrating different aspects of the organization—architecture, technical details, business requirements, and team dynamics—ensuring that everything works together seamlessly. This role is vital in large organizations and complex projects where cohesion and coordination are paramount to success. The section [Thinking Like an Architect: Architects as Superglue](#) provides more details.

Balancing curiosity, doubt, vision, and skepticism is essential for driving sustainable innovation and change in organizations. Architects must constantly reflect on how well they balance these forces. In doing so, they can ensure that their efforts are driven by a healthy combination of exploration, critical validation, strategic guidance, and cautious realism—all crucial to achieving lasting success in any organizational change. The section [Thinking Like an Architect: Balancing Curiosity, Doubt, Vision, and Skepticism](#) provides more details.

The ideal career path for architects is rooted in a strong engineering background. This foundation provides the technical proficiency necessary for advanced architectural roles. Moving from an engineering role to an architecture role involves broadening scope, increasing diversity, and developing strong communication and influencer skills. Architecture roles can lead to tech leadership positions such as Engineering Director or Chief Technology Officer (CTO), leveraging strategic vision, decision-making, and leadership skills.

The section [Architects’ Career Paths](#) provides more details.

# 12: Building Skills

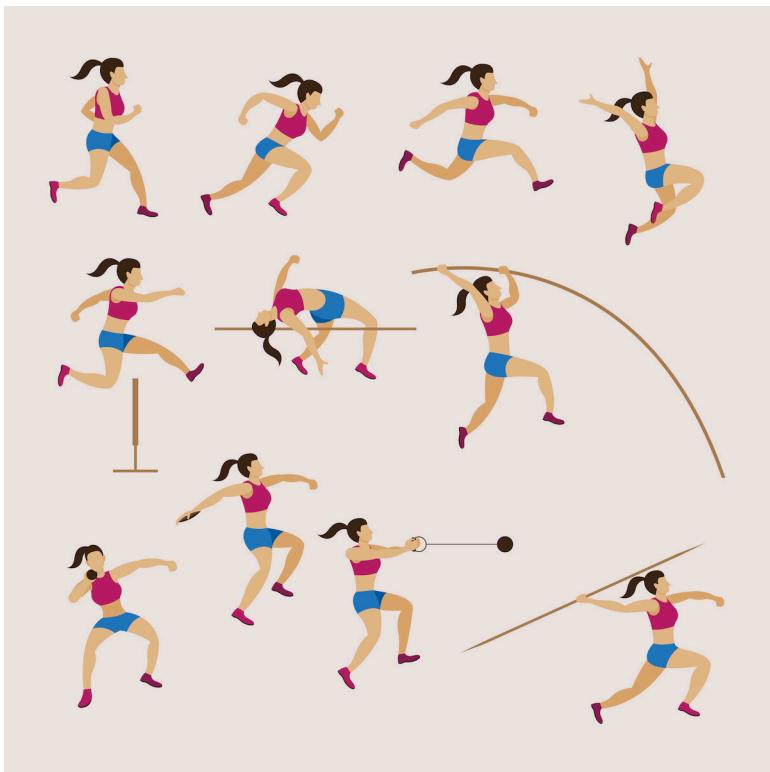


image by muchmania from istock

**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.

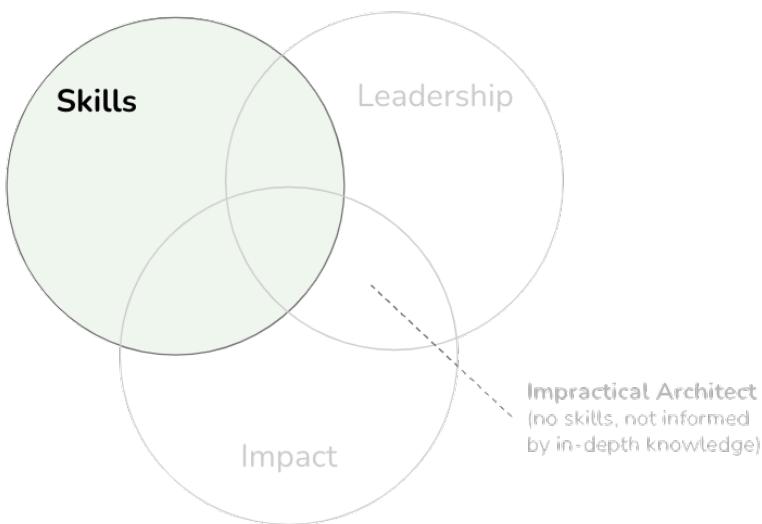
**KEY POINTS:**

- An architect's typical skillset includes hard (technical) skills, soft (people & social) skills, product development, business skills, and decision-making skills.
- Hard (technical) skills are essential for designing, implementing, and maintaining an organization's technology landscape.
- Soft skills are integral to social architecture, enabling individuals to navigate and contribute to these social systems effectively.

- Product development knowledge is the bridge that helps architects align technical solutions with customer needs and business objectives.
- Business domain knowledge is not just useful but essential for architects to create solutions that deliver real value.

- Decision-making skills ensure that architectural decisions are sound, sustainable, and aligned with long-term strategic objectives.

Architects must possess a comprehensive skill set to manage the complexities of modern IT environments effectively (Figure 1). By skills, I mean not only having relevant knowledge but also the ability to apply that knowledge in practical situations. These skills encompass a blend of technical expertise, communication proficiency, and influence capabilities, ensuring architects can navigate technological challenges and organizational dynamics.



**Figure 1:** Skills are one of the three main elements of being an architect (skills, impact, leadership). Lack of skills leads to impractical decisions that are not informed by in-depth knowledge.

Core skills of architects include:

- **Technical Skills:** Architects need a robust foundation in new and legacy technology stacks. This foundation includes proficiency in topics like cloud architecture, containerization technologies like Kubernetes, and a deep understanding of various programming languages and frameworks. Staying updated with emerging technologies and industry trends is essential for making informed decisions that align with organizational goals.
- **Communication Skills:** Effective communication is crucial for architects. They must articulate complex technical concepts in a way that is understandable to non-technical stakeholders. These skills involve clear verbal and written communication, active listening, and the ability to tailor messages to different audiences. Strong communication skills help bridge the gap between technical teams and business units, fostering collaboration and mutual understanding.
- **Influence Skills:** Architects often need to persuade and influence others, including stakeholders, developers, and executives. This re-

quires negotiation skills, the ability to build consensus, and the credibility to advocate for necessary changes. Influence skills enable architects to drive strategic initiatives and ensure that architectural decisions are implemented effectively across the organization.

- **Product Development Skills:** Architects must understand the product development lifecycle. This understanding should include knowledge of product development methodologies, user experience design, and product management principles. Architects should be able to contribute to product strategy, ensuring that technical solutions align with customer needs and business objectives.
- **Business Domain Knowledge:** A deep understanding of the business domain is essential for architects to create solutions that deliver real value. This knowledge includes industry-specific regulations, market trends, and competitive landscape. Business acumen helps architects align technical initiatives with business goals, driving growth and innovation.
- **Decision-Making Skills:** Architects are often required to make critical decisions that impact the entire organization. This involves assessing risks, evaluating trade-offs, and making informed choices based on data and experience. Strong decision-making skills ensure that architectural decisions are sound, sustainable, and aligned with long-term strategic objectives.

## 12.1: Technical Skills

Hard or technical skills are the abilities and knowledge needed for designing, implementing, and maintaining various aspects of an organization's technology landscape.



image by rgstudio from istock

Some typical hard skills that architects need in their work include:

- **System design**<sup>1</sup>: This involves 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. System design skills ensure the architect can envision and structure systems effectively to meet organizational goals.
- **Engineering processes**<sup>2</sup>: An in-depth understanding of engineering processes, including the software development life cycle, Agile development, DevOps, and continuous delivery, is crucial. Architects must ensure their systems are developed efficiently and effec-

<sup>1</sup><https://blog.pragmaticengineer.com/system-design-interview-an-insiders-guide-review/>

<sup>2</sup><https://obren.io/tools/catalogs/?id=design-tactics-high-performing-technology-organizations>

tively, adhering to best practices and methodologies that enhance software development productivity and quality.

- **Design patterns<sup>3</sup> and tactics<sup>4</sup>:** Familiarity with design patterns and tactics such as Cloud Design Patterns, Model-View-Controller (MVC), Service-Oriented Architecture (SOA), and Microservices is essential. These patterns help architects design modular, scalable, and maintainable systems, providing solutions to common design problems and ensuring that the systems can evolve and adapt over time.
- **Security and privacy by design<sup>5</sup>:** With cybersecurity's increasing importance, architects need a deep understanding of security and privacy best practices. They must design secure and compliant systems with data protection regulations, incorporating security measures at every stage of the design process to protect sensitive information and mitigate risks.
- **System optimizations<sup>6</sup>:** Knowledge of optimizing systems for performance and scalability is critical. Architects should be adept at using tools and techniques for profiling and tuning systems to achieve optimal performance, ensuring that the systems can handle increased load and provide a seamless user experience.
- **Source code structures and maintainability<sup>7</sup>:** A good understanding of software engineering principles such as clean code, code maintainability, and refactoring is important. Architects should design systems that are easy to maintain and modify, promoting long-term sustainability and reducing the technical debt that can accumulate over time.
- **Reliability and stability (anti)patterns<sup>8</sup> and tactics<sup>9</sup>:** Understanding typical reliability and stability issues in complex systems is crucial. Architects should identify and address potential problems using patterns and tactics such as redundancy, failover, and graceful degradation to ensure that systems remain stable and reliable under varying conditions.

---

<sup>3</sup>[https://obren.io/tools?tag=design\\_patterns](https://obren.io/tools?tag=design_patterns)

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

<sup>5</sup><https://obren.io/tools?tag=security>

<sup>6</sup><https://obren.io/tools/catalogs/?id=design-tactics-sig-performance>

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

<sup>8</sup><https://obren.io/tools/catalogs/?id=releaseit-stability-awareness>

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

- **Usability**<sup>10</sup>: A good understanding of usability principles is necessary for designing systems that are easy to use and provide a good user experience. Architects must ensure that their designs are user-friendly, intuitive, and accessible, enhancing end-users' overall satisfaction and productivity.

By mastering these skills, technical architects can effectively contribute to the success of their organizations, ensuring that technology solutions are well-designed, secure, efficient, and user-centric.

The section [Appendix](#) provides some pointers for resources to build your technical skills.

---

<sup>10</sup><https://obren.io/tools?q=usability>

## 12.2: 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.



image by peopleimages from istock

Critical soft skills include:

- **Communication skills<sup>11</sup>**, including **written<sup>12</sup>**, **visual<sup>13</sup>**, verbal (presentation), and listening skills: Effective communication involves expressing oneself clearly and understanding and empathizing with others. These skills are essential for building and maintaining relationships, as well as for conveying ideas and facilitating discussions. Written communication ensures clear and concise messages, visual communication enhances understanding through diagrams and presentations, verbal communication is critical in delivering impactful presentations, and active listening fosters understanding and collaboration.
- **Networking and collaboration skills<sup>14</sup>**: Networking involves building and maintaining diverse professional connections. Collaboration skills encompass working effectively with others, regardless of their role or seniority. These skills include partnering with peers, junior and senior colleagues, managers, and executives to achieve common goals. Effective networking opens doors to new opportunities and resources, while strong collaboration skills ensure that team efforts are synergistic and productive.
- **Organizational and time management skills<sup>15</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. Mastering these skills helps individuals stay on top of their responsibilities, reduces stress, and enhances overall productivity.
- **Analytical, strategic thinking, and problem-solving skills<sup>16</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

---

<sup>11</sup><https://obren.io/tools?tag=consultancy>

<sup>12</sup><https://obren.io/tools/sowhat/>

<sup>13</sup><https://obren.io/tools?tag=visuals>

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

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

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

capitalizing on unique opportunities and creating organizational value. By honing these abilities, individuals can contribute to innovative solutions, drive strategic initiatives, and confidently navigate complexities.

By developing these soft skills, individuals can significantly enhance their ability to function effectively within organizations. These skills foster environments where collaboration, efficiency, and innovation thrive, leading to personal success and contributing to the overall health and productivity of the organizations and communities they are a part of.

The [Appendix](#) provides some pointers for resources to build your soft skills.

## 12.3: 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.



image by tirachard from istock

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. It's a creative phase where innovation is encouraged, and a broad range of ideas are considered.
- **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. This step helps understand the market landscape and identify the product's unique selling points.

- **Product Design and Development:** This phase involves turning the validated concept into a detailed product design. It includes creating prototypes, testing, and refining the product design based on feedback and performance metrics. Collaboration between design and engineering teams is crucial to ensure the product is functional, aesthetically pleasing, and manufacturable.
- **Testing and Validation:** The product undergoes rigorous testing to ensure it meets quality standards and performs as expected. This stage includes usability testing, user performance testing, and sometimes beta testing with a limited audience. Feedback collected during this phase is used to make necessary adjustments and improvements.
- **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. A successful launch plan ensures the product reaches the intended audience effectively.
- **Post-Launch Evaluation and Iteration:** After the product launch, it is important to monitor its performance in the market. This activity includes collecting customer feedback, analyzing sales data, and assessing overall market reception. Continuous improvement is essential; iterations based on this feedback help refine the product and address any issues.

Understanding the product development process is not just beneficial, but essential for architects. It requires a multidisciplinary approach involving teams from various functions such as product management, design, engineering, and marketing. This understanding is crucial for architects, as it helps them align their technical designs with the overall product strategy, ensuring that a product is not only technically sound but also meets market demands and customer expectations. The process aims to create innovative, desirable, and commercially viable products that meet customer needs and provide a competitive advantage in the market.

## 12.4: 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.



image by azmanl from istock

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. This broad knowledge base helps architects to see the bigger picture, understand how their technical decisions impact the business, and ensure their designs support the organization's strategic objectives. *The Personal MBA*<sup>17</sup> book is a valuable resource for familiarizing oneself with such concepts, offering insights into essential business principles and practices.

---

<sup>17</sup><https://personalmba.com/>

- **Specific Business Domains<sup>18</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 includes 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. For example, an architect in the healthcare industry needs to understand healthcare regulations, patient care standards, and the evolving needs of healthcare providers and patients.
- **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. Practical business analysis ensures that the solutions architects design are aligned with business goals and deliver tangible value. This process often includes conducting interviews, facilitating workshops, and using techniques such as SWOT analysis (Strengths, Weaknesses, Opportunities, Threats) and business model canvases to capture and prioritize requirements.
- **Stakeholder Management:** Communication and relationship management with stakeholders is crucial. Architects must understand the interests and concerns of different stakeholders, including executives, managers, employees, and customers. This understanding helps gain buy-in for architectural decisions and ensure that the solutions meet stakeholder needs.
- **Project Management:** Basic project management skills enable architects to oversee the implementation of their designs, ensuring projects are completed on time, within scope, and budget. Understanding project management methodologies can help architects work effectively with project managers and development teams.
- **Finance:** Architects should understand basic financial principles, including budgeting, cost-benefit analysis, return on investment (ROI), and "Earnings Before Interest, Taxes, Depreciation, and

---

<sup>18</sup>[https://obren.io/tools?tag=domain\\_models](https://obren.io/tools?tag=domain_models)

Amortization” (EBITDA). This knowledge helps them make cost-effective decisions and justify the financial viability of proposed solutions.

- **Strategic Thinking:** Architects need to think strategically and understand how their work supports the organization’s long-term goals. This thinking includes aligning technology initiatives with business strategy, identifying opportunities for innovation, and anticipating future business needs.
- **Change Management:** Implementing new systems and architectures often involves significant organizational change. Architects should be familiar with change management principles to help facilitate smooth transitions, manage resistance, and successfully adopt new technologies and processes.

By mastering these business skills, architects can ensure that their technical solutions are innovative, efficient, and aligned with their organizations’ broader business goals and strategies. This holistic approach enables architects to contribute more effectively to organizational success and drive meaningful business outcomes.

## 12.5: Decision-Making Skills

Architects' work always requires pragmatic decision-making. Decisions are the steering wheel of organizations, and architects who are not involved in key decisions will have a limited impact on the organization.



image by gorodenkoff from istock

Architects will have three roles concerning decision-making:

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

Essential decision-making skills include:

- **Understanding that a decision is an irrevocable allocation of resources:** This involves recognizing that every decision commits resources such as money, human effort, time, physical actions, and opportunities. Architects must consider the long-term implications of these allocations, ensuring that resources are used efficiently and effectively to achieve organizational goals.
- **Know the basics of decision intelligence:** Decision intelligence is the discipline of turning information into better action. It involves leveraging data, analytics, and cognitive science to enhance decision-making processes. By understanding and applying

decision intelligence, architects can make more informed, evidence-based decisions that lead to better outcomes.

- **Understand key problems of poor decision-making, such as the outcome bias:** Outcome bias occurs when decisions are judged based on their results rather than the quality of the decision-making process itself. Architects need to recognize and mitigate this bias by focusing on the decision-making process, ensuring that decisions are made systematically and rationally, regardless of the outcomes.
- **Know when to use intuition:** Intuition can be a valuable tool in decision-making, especially when data is limited or time is of the essence. Architects should understand when to rely on their intuition and when to seek additional information.
- **Understand that there is no such thing as not making a decision:** Delaying or postponing a decision is a decision with its own set of consequences. Architects must know the implications of inaction and understand that deferring decisions can lead to missed opportunities, increased risks, and reduced organizational agility.
- **Risk Assessment:** Understanding and assessing risks is crucial for making informed decisions. Architects should evaluate potential risks associated with different options and develop strategies to mitigate these risks.
- **Collaborative Decision-Making:** Involving relevant stakeholders in decision-making ensures that diverse perspectives are considered, leading to more robust and well-rounded decisions. Architects should facilitate collaboration and consensus-building among team members.
- **Ethical Considerations:** Architects must ensure their decisions align with ethical standards and organizational values. These actions involve considering the broader impact of decisions on stakeholders, society, and the environment.
- **Adaptability and Flexibility:** In a rapidly changing business environment, architects must be adaptable and open to revisiting and revising decisions as new information becomes available. Flexibility allows for continuous improvement and responsiveness to emerging challenges and opportunities.

Architects can enhance their ability to influence and drive organizational success by developing these decision-making skills. Whether acting as decision-makers, advisors, or evaluators, architects are critical in steering their organizations toward strategic goals and sustainable growth.

## 12.6: Integrating Skills for Success

Architects must integrate these diverse skills into a cohesive approach to be truly effective. They must be lifelong learners, continuously updating their knowledge and adapting to new challenges.



image by colorsandia from istock

Here's how these skills come together in practice:

- **Technical Proficiency:** Enables architects to design robust, scalable solutions that leverage the latest technologies.
- **Effective Communication:** Ensures that architectural visions are conveyed and understood across all levels of the organization.
- **Influence and Persuasion:** Help garner support for architectural initiatives and drive change.
- **Product Insight:** Aligns technical solutions with customer needs and market demands.
- **Business Understanding:** Ensures that architectural decisions contribute to the organization's strategic objectives.
- **Decisive Leadership:** Guides the organization through complex technological landscapes with confidence and clarity.

Architects can effectively manage the interplay between technology, business, and organizational dynamics by developing and honing these skills. This comprehensive skill set enhances their ability to design innovative solutions and ensures they can lead teams and influence stakeholders to achieve a cohesive and successful architectural vision.

## 12.7: To Probe Further

- Appendix: Bookshelf<sup>19</sup>
- Old Books that Every Architect Should Read<sup>20</sup>, by Gregor Hohpe, 2024
- Back from the engine room<sup>21</sup>, by Gregor Hohpe, 2023
- Debugging Architects<sup>22</sup>, by Gregor Hohpe, 2021

---

<sup>19</sup>[bookshelf](#)

<sup>20</sup><https://architectelevator.com/architecture/classic-architecture-books/>

<sup>21</sup><https://architectelevator.com/transformation/debugging-architect/>

<sup>22</sup><https://architectelevator.com/architecture/engine-room/>

## 12.8: 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?*

## 13: Making Impact



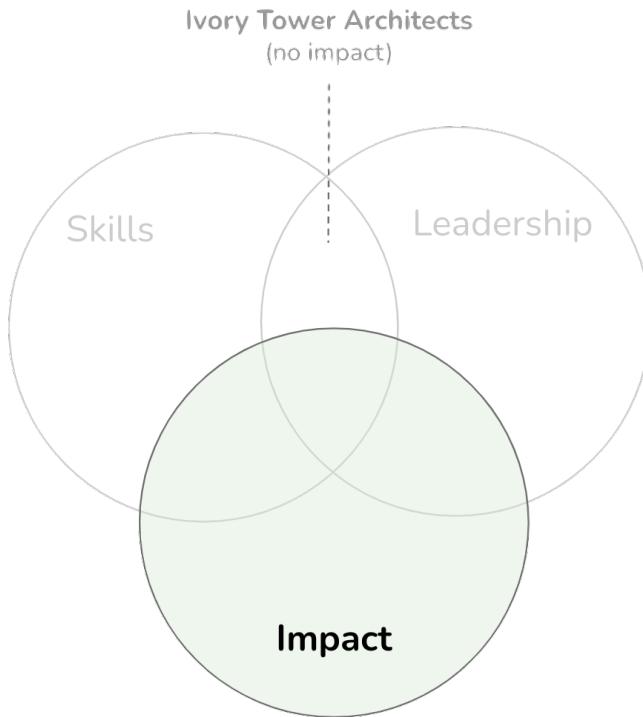
image by tuiphotoengineer from istock

**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** (Figure 1).



**Figure 1:** Impact is one of the three main elements of being an architect (skills, impact, leadership). Leadership without impact lacks foundation and may signal that you have become an ivory tower architect with a weak

*relation to reality.*

Architects' contributions are typically assessed through the following key areas:

- **Identifying, Tackling, and Delivering on Strategic Problems:** Architects are responsible for recognizing and addressing strategic issues at both the organizational and area-specific levels (such as domains or technical areas). They can facilitate aligning work with the organization's broader strategic objectives, ensuring that efforts are prioritized to support global goals. By tackling these strategic problems, architects contribute to the organization's overall direction and success, providing solutions that align with long-term visions and targets.
- **Having a Deep and Broad Influence:** Architects must have a profound and extensive impact on their specific domain, product, or technology area. This influence requires them to delve deeply into particular critical issues, providing targeted solutions that address pressing challenges. Simultaneously, they need to maintain a broad perspective, creating value by leveraging their solutions across multiple teams and projects. This dual focus ensures that their influence is both concrete and widespread, enhancing the overall efficiency and effectiveness of the organization.
- **Delivering Solutions that Few Others Can:** Architects are often tasked with providing solutions that are beyond the reach of others, either through their direct efforts or by orchestrating large-scale group endeavors. Their unique combination of hard technical skills and soft skills in strategy, execution, and people management enables them to navigate complex challenges and drive significant progress. By leveraging these skills, architects can move the organization forward, ensuring that the solutions they deliver are both innovative and implementable, fostering growth and development across the board.

Architects make a significant impact by identifying and solving strategic problems, exerting deep and broad influence across domains, and delivering unique solutions through a blend of technical expertise and strategic leadership. This multifaceted approach ensures that their work is not only aligned with organizational objectives but also drives substantial progress and innovation.

## 13.1: Pillars of Impact

Architects must possess strong technical, people, and business skills, which are ideally developed through extensive practice and experience. Building on this robust foundation, architects need to cultivate specific competencies that enable them to leverage their experiences and abilities to **positively impact organizational performance**. As architects advance in their careers, their competency development should be increasingly driven by the desired impact on the organization rather than solely on acquiring new skills.

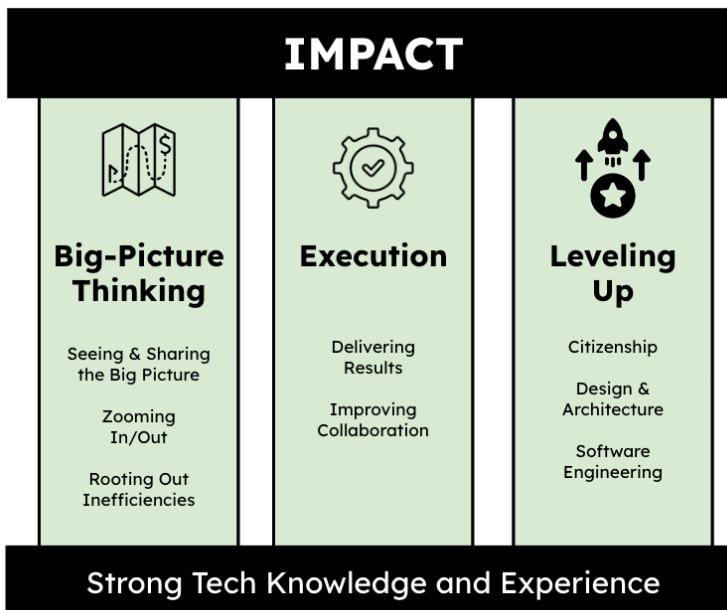
I typically coach architects within the framework of concrete activities, focusing on real-world challenges and guiding their development through hands-on involvement in appropriate actions. This practical approach helps tailor their skill development to address specific challenges and achieve the expected impact in practice.

To develop a structured approach to teach architects how to make an impact, I draw inspiration from Staff Engineering roles. 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> provide valuable insights into defining the responsibilities and expectations of architects.

---

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

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



*Figure 2: Key competencies of architects. Inspired by The Staff Engineer's Path by Tanya Reilly.*

Inspired by *The Staff Engineer's Path* by Tanya Reilly (Figure 2), I categorize the competencies that enable architects to make an impact into three groups:

- **Big-picture Thinking:** Architects need to develop the ability to see the broader context of their work. This type of impact involves understanding how their architectural decisions align with organizational goals, market trends, and technological advancements. Big-picture thinking allows architects to foresee potential challenges and opportunities, ensuring their designs are future-proof and scalable. This competency involves strategic planning, vision setting, and the ability to articulate how technical solutions contribute to overall business objectives.
- **Execution:** Execution is about turning ideas into reality. Architects must help teams complete projects efficiently and effectively. This type of impact involves project management skills, the ability to coordinate with cross-functional teams, and a deep understanding of

the technical intricacies of bringing a project to fruition. Architects must also be capable of troubleshooting issues, adapting to changes, and ensuring that the final deliverables meet the required standards and expectations.

- **Leveling Up:** Leveling up refers to the continuous improvement of both the architect's skills and the capabilities of their teams. Architects must be committed to personal growth and the development of those around them. This type of impact involves mentoring junior team members, fostering a culture of learning, and staying abreast of the latest technological advancements and industry best practices. Architects should also focus on enhancing the overall skill set of their teams, ensuring that knowledge and expertise are shared and that the team collectively grows stronger and more capable.

By focusing on these three competency groups—big-picture thinking, execution, and leveling up—architects can significantly enhance their impact on organizational performance. This structured approach to competency development can ensure that architects are skilled practitioners and strategic leaders capable of driving meaningful change and innovation within their organizations.

### **13.1.1: Big-Picture Thinking**

Architects are often the only individuals within an organization who possess a “helicopter view,” allowing them to oversee vast domains and anticipate the broader consequences of decisions.



image by guvendemir from istock

This unique perspective positions them as crucial big-picture thinkers who can contribute to the organization in several significant ways:

- **Identifying High-Leverage Points for Maximum Impact:** With their ability to see the big picture, architects can pinpoint areas within the organization where small changes or strategic investments can yield substantial benefits. This capability allows them to recommend and implement solutions that maximize impact, ensuring resources are allocated efficiently and effectively. By focusing on high-leverage points, architects can drive significant improvements in performance and productivity across the organization.
- **Helping Others to See the Big Picture:** Architects can facilitate a broader understanding of the organizational landscape among their colleagues. By helping others see the big picture, they enable more informed decision-making at all levels. They can create tools and frameworks, such as a comprehensive [Lightweight Architectural Analytics](#), that promote big-picture thinking. These tools help team members understand how their work fits into the larger organizational context and the impact of their contributions. Through workshops, presentations, and collaborative tools, architects can disseminate their vision, fostering a culture of strategic awareness.
- **Zooming In and Out:** Good architects have the rare ability to both maintain a strategic overview and delve into implementation

details when necessary. This flexibility allows them to bridge the gap between high-level strategy and on-the-ground execution. They can provide strategic guidance while also engaging deeply with technical teams to ensure that the finer details align with broader organizational goals. This dual capability ensures that strategic initiatives are feasible and that practical implementations stay aligned with the overall vision.

- **Rooting Out Inefficiencies:** Using their big-picture perspective, architects are adept at identifying and eliminating inefficiencies within organizational processes and systems. They can lead the adoption of new technologies and processes that enhance the efficiency and effectiveness of multiple teams. By doing so, architects can help streamline operations, reduce redundancies, and improve overall performance. This proactive approach to efficiency ensures that the organization remains competitive and can adapt to changing market conditions and technological advancements.

Architects play a pivotal role in organizations by leveraging their helicopter view to identify high-leverage points, help others understand the big picture, seamlessly transition between strategic and detailed thinking, and root out inefficiencies. Their ability to see and act on the broader context makes them invaluable in driving organizational success and fostering a culture of strategic innovation.

### 13.1.2: Execution

As execution-focused practitioners, architects must not only deliver tangible results but also enhance collaboration across the organization. Their ability to combine technical expertise with **pragmatic approaches** and foster **effective teamwork** is crucial for achieving these goals.



image by laylabird from istock

Here's how architects can excel in these areas:

#### **13.1.2.1: Delivering Results with Pragmatism**

Architects must blend their technical expertise with practical approaches to ensure their solutions are **impactful and feasible**. By prioritizing practicality and feasibility, architects can ensure their contributions lead to tangible improvements and successful outcomes:

- **Create Meaningful Solutions:** Architects should focus on developing solutions that are practical and impactful rather than getting caught up in theoretical ideals and models. These solutions must address real-world problems and be implementable within the current organizational context. By prioritizing practicality, architects ensure that their contributions lead to tangible improvements and benefits.
- **Break Down Complex Problems:** To deliver impactful results, architects must be skilled at deconstructing complex issues into manageable components. This approach allows teams to tackle problems step-by-step, reducing overwhelm and increasing the likelihood of successful outcomes. By using this divide-and-conquer

approach, architects help ensure that projects progress smoothly and deliver the desired impact.

- **Craft Pragmatic Plans:** When planning projects, architects must consider various constraints, including technical limitations, logistical challenges, and organizational dynamics. Pragmatic planning involves balancing ideal solutions with what is feasible given these constraints. This approach ensures that plans are realistic, actionable, and more likely to succeed in the real world.

### **13.1.2.2: Enhancing Collaboration**

Architects can play a crucial role in fostering **effective collaboration** and creating alignment within their teams and across the broader organization. By engaging meaningfully with various teams and departments, architects build trust and enhance execution speed, leading to better overall productivity and execution.

- **Creating Alignment:** Architects can be key in creating alignment within their teams and the broader organization. This role involves ensuring all stakeholders understand the strategic objectives and how their work contributes to these goals. Clear communication and shared understanding help align efforts and focus resources on common objectives.
- **Improving Collaboration:** Architects can be critical catalysts for improving collaboration within their teams and across different groups in the organization. This role facilitates communication, coordinates efforts, and resolves conflicts to ensure smooth cooperation. By fostering a collaborative environment, architects can help teams work more effectively together, leading to better execution and faster results.
- **Collaborating Meaningfully Across Groups:** Architects should actively engage with various teams and departments to build trust and improve execution speed. Meaningful collaboration involves understanding the needs and perspectives of different groups and finding ways to integrate their efforts. Building trust through consistent and open communication can help to break down silos and enhances overall productivity.

As execution-focused practitioners, architects must blend their technical skills with a pragmatic approach to deliver meaningful solutions and break down complex problems. They must also play a vital role in creating alignment and improving collaboration within and across teams. By doing so, they not only achieve impactful results but also enhance the overall efficiency and effectiveness of the organization.

### 13.1.3: Leveling Up

Architects are often seen as leaders and role models who can help organizations raise the bar on both technical and cultural fronts.



image by sanjeli from istock

This role I categorize into three key areas: citizenship, design and architecture, and software engineering.

#### 13.1.3.1: Citizenship

Architects should look beyond their immediate responsibilities and work to elevate practices and behaviors across their organizations:

- **Contribute to the broader technical community** through tech talks, education, publications, and open-source projects. This

involvement helps spread knowledge and innovation.

- **Extend their influence** beyond their organization to make an impact on the industry at large, sharing insights and best practices that benefit a wider audience.
- **Lead efforts** to solve **significant problems** in their areas, demonstrating leadership and commitment to continuous improvement.
- **Elevate the engineering culture** within the company, fostering an environment of excellence and continuous learning.

### **13.1.3.2: Design and Architecture**

As leading authorities on systematic and strategic design, architects play a critical role in shaping the architecture of their organizations:

- Leverage their deep domain knowledge to **improve the definition of best practices** in design and architecture, ensuring that standards are maintained and enhanced over time.
- **Identify and solve systemic architectural problems** by quickly recognizing issues and articulating possible solutions. Their ability to address these challenges ensures system stability and scalability, ensuring high-quality overall architecture.

### **13.1.3.3: Software Engineering**

Architects also need to stay deeply connected to software engineering practices, leveraging their experience to enhance technical execution:

- **Promote and demonstrate best-in-class practices** in coding, documentation, testing, and monitoring. By setting high standards, they can ensure that quality and efficiency are prioritized.
- Solve **challenging technical and execution problems** that few others can, using their expertise to tackle issues that require advanced skills and innovative thinking. Architects' influence can motivate others to develop new advanced skills.

By excelling in these areas, architects can significantly contribute to raising their organizations' technical and cultural standards, leading by example, and driving continuous improvement across all levels.

## 13.2: Questions to Consider

- *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?*

# 14: Leadership Traits



image by niserin from istock

**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, I see the following behavioral traits as crucial for architects: communication, judgment, impact, inclusion, selflessness, courage, integrity, curiosity, innovation, and passion.

My approach to architecture leadership draws inspiration from two standout sources: **David Marquet's<sup>1</sup>** leadership principles, articulated in his book "**Turn the Ship Around!**<sup>2</sup>" and "**Leadership Is Language**<sup>3</sup>", and Netflix's valued behaviors. Marquet's ideas are about empowering team members, providing clarity, decentralizing decision-making, striving for continuous improvement, and practicing servant leadership. Meanwhile, **Netflix's valued behaviors<sup>4</sup>** offers a compact masterclass in coaching and developing people's leadership traits. Together, these resources form a robust framework for leading with insight and influence in the IT architecture (Figure 1).

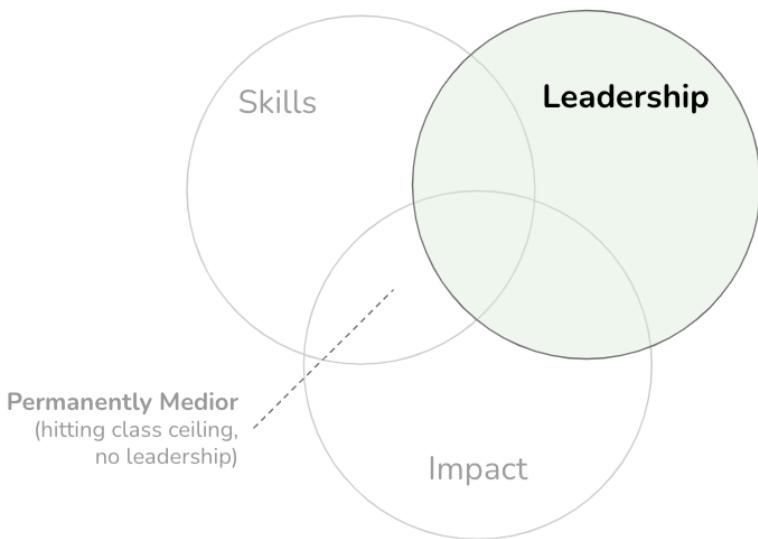
---

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

<sup>2</sup><https://davidmarquet.com/turn-the-ship-around-book/>

<sup>3</sup><https://davidmarquet.com/leadership-is-language-book/>

<sup>4</sup><https://jobs.netflix.com/culture>



**Figure 1:** Leadership is one of the three main elements of being an architect (skills, impact, leadership). 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.

## 14.1: 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 should 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 caiaimage /martin barraud from 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 2). 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 can expect to **take the initiative and lead efforts**. In addition, this phase captures the desired interaction of the teams with whom architects work, expecting teams to take the initiative and share their intentions, which architects can support and level up.

The **Leader-Leader model** can and has been applied effectively in software engineering and IT architecture. Here are some examples:

- **Empowering Teams:** In Agile environments, authority is often distributed among team members, embodying the Leader-Leader model. Instead of a project manager or lead developer dictating

every detail, team members collaborate closely, take ownership of specific tasks, and make decisions within their areas of expertise. This approach empowers developers, testers, and other team members to contribute actively, promoting creativity and innovation.

- **Cross-functional Collaboration:** In a DevOps environment, the boundaries between development and operations are blurred. Teams are empowered to take end-to-end ownership of the product lifecycle, from development to deployment and monitoring. This shared responsibility encourages collaboration and reduces the reliance on a single decision-maker, aligning closely with the Leader-Leader model.
- **Continuous Improvement:** DevOps emphasizes continuous integration, continuous deployment, and continuous feedback. Teams are encouraged to experiment, learn from failures, and improve processes. Leaders in DevOps act as coaches, guiding teams to identify opportunities for improvement and innovation, rather than directing every action.
- **Mentoring and Coaching:** Senior architects often take on the role of mentors and coaches rather than top-down decision-makers. They guide less experienced team members through complex architectural challenges, fostering a culture of learning and development. This approach helps build a strong, capable architecture team that can independently contribute to the organization's success.
- **Community-Driven Leadership:** Open source projects often operate under the Leader-Leader model. Contributors from around the world collaborate on projects, and decision-making is distributed across the community. Leaders in open source projects often emerge organically, based on their contributions and the respect they earn from others, rather than through formal authority.
- **Collaborative Problem-Solving:** In open-source communities, problems are solved through collaboration and shared knowledge. Contributors propose solutions, review each other's code, and collectively decide on the best approaches. This inclusive and collaborative environment encourages innovation and creativity, reflecting the principles of the Leader-Leader model.
- **Decentralized Control:** In a microservices architecture, different teams manage different services independently. Each team is responsible for the full lifecycle of its services, from design to deployment and maintenance. This autonomy aligns with the

Leader-Leader model, where each team is empowered to make the best decisions for their service while collaborating with other teams to ensure the overall system's success.

- **Service Ownership:** Teams practicing microservices architecture are encouraged to adopt a “you build it, you run it” philosophy. This ownership means they develop the services and operate and maintain them in production. This ownership fosters accountability and drives teams to improve their services continuously.

In all these examples, the Leader-Leader model promotes a collaborative, inclusive, and empowering work environment. Individuals are encouraged to take ownership of their work and contribute to the collective success of the team or organization. This approach enhances productivity and fosters innovation, creativity, and continuous improvement.

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.

*Figure 2: Leadership language. Based on Intent-Based Leadership, by David Marquet.*

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

The [Netflix overview of their valued behaviors](#)<sup>5</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.



image by chaiyon021/martin barraud from istock

### 14.2.1: 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

---

<sup>5</sup><https://jobs.netflix.com/culture>

- You **adapt your communication style** to work well with people from around the world who may not share your native language

### **14.2.2: 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**
- 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

### **14.2.3: Impact**

As discussed in the [Impact](#) section, the architect's work 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**

### **14.2.4: 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

#### **14.2.5: 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

#### **14.2.6: 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

### 14.2.7: 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

### 14.2.8: 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

### 14.2.9: 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**

#### **14.2.10: 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**

### 14.3: 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?

# 15: Thinking Like an Architect: Architects as Superglue



image by pagadesign from istock

**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.

To succeed as an IT architect, you need the skills and the right **mindset**—a set of attitudes, beliefs, and mental frameworks that shape how you perceive and respond to situations. Regarding their role in organizations, I have found the “**superglue**” **metaphor** to be an effective way to put architects into the mindset that sets them up for success and making an impact.

The concept of “superglue” people in IT organizations emphasizes the critical role of those who act as the **binding force** across various facets of an organization. This idea, championed by [Adam Bar-Niv and Amir Shenhav from Intel](#)<sup>1</sup>, and echoed by [Tanya Reilly](#)<sup>2</sup>, underscores the need for individuals who excel in more than just technical skills. Gregor Hohpe similarly describes modern architects’ primary role as gluing different organizational functions by riding the [Architect Elevator](#)<sup>3</sup> from the penthouse, where the business strategy is set, to the engine room, where engineers implement enabling components, services and systems.

Architects acting as “superglue” should aim to hold together architecture, technical details, business needs, and people within large organizations or complex projects. They should function as the “**organizational connective tissue**,” with their primary strength being the ability to relate technical issues to broader organizational and business contexts.

---

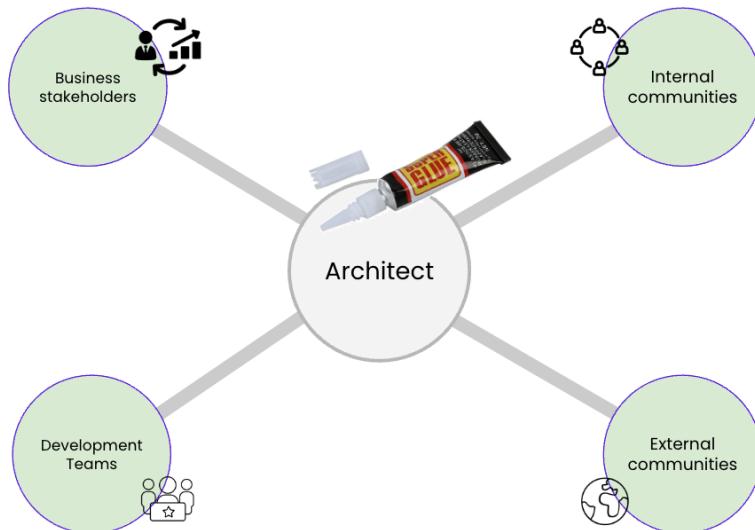
<sup>1</sup><https://saturn2016.sched.com/event/63m9/cant-find-superheroes-to-help-you-out-of-a-crisis-how-about-some-architecture-and-lots-of-superglue>

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

<sup>3</sup><https://architectelevator.com/>

## 15.1: Superglue Architects

While technical expertise is indispensable, what should set superglue architects apart from technical specialists is their exceptional **relational skills**. They must communicate effectively, negotiate, and influence various stakeholders to achieve alignment and drive projects forward. This unique blend of technical and interpersonal skills is the reason for having architects and makes them invaluable in maintaining the organization's coherence and efficiency.



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

Figure 1 captures the **superglue metaphor** for architects, showing how they, like superglue, can bond various parts of the organization:

- **Developer Whisperers:** Architects should work closely with developers, understanding their challenges and ensuring architectural decisions enhance development efficiency and effectiveness.
- **Tech-to-Business Translators:** They should decode technical jargon into business terms, helping stakeholders grasp the value and implications of technical decisions.

- **Cross-Functional Diplomats:** By engaging with operations, marketing, and finance, architects can ensure technical solutions are practical, viable, and aligned with organizational strategies.
- **Community Connectors:** Active engagement within internal communities keeps architects informed and contributes to the collective knowledge pool.
- **Industry Influencers:** By being visible externally, architects can learn from and influence the broader industry, bringing in fresh perspectives and establishing their organization as a thought leader.

Superglue architects can play a unique and essential role in the smooth operation of large and complex IT organizations. Unlike superheroes who save the day with dramatic rescues, superglue architects ensure continuous, smooth operation by connecting the dots across various organizational aspects.

IT architects should focus on holding everything together, ensuring their organizations' stability, coherence, and progress. Think of them as the organizational equivalent of duct tape—versatile, indispensable, and always ready to fix the seemingly unfixable.

## 15.2: Supergluing in Action #1: Aligning Organization

Tensions among technology, product, organization, and business functions can slow down progress, lead to poor decisions, introduce complexity, and cause missed opportunities. Acting as superglue, architects can mitigate these issues by fostering better communication and alignment among these elements. The goal is **not to create new barriers** but to bring these functions closer together, ensuring a cohesive and efficient operation.

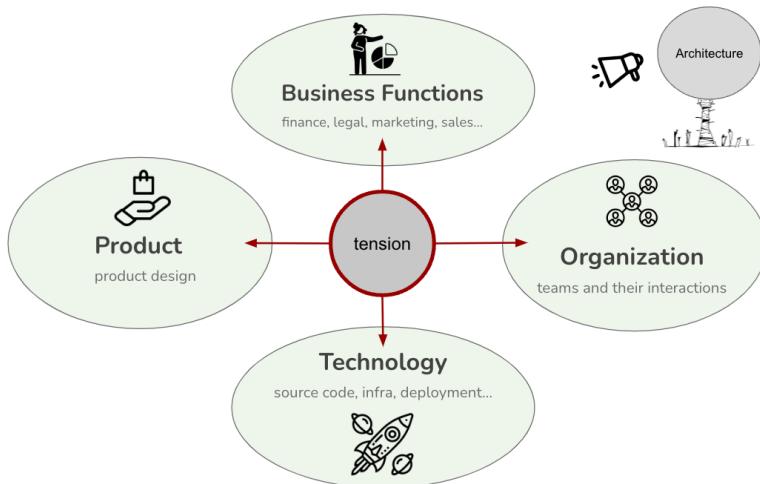


image by flamingoimages from istock

The primary value of superglue architects in complex organizations lies in their uncanny ability to align business, product, technology, and organizational functions.

### 15.2.1: Tensions

Technology, product, organization, and business functions face specific challenges and change at different rates. Ideally, these structures should **evolve simultaneously**, staying in perfect sync like a well-rehearsed dance troupe. However, reality often resembles a **poorly synchronized** flash mob, full of tension and missteps, as illustrated in Figure 2.



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

Tension between technology, product, organization, and business functions can significantly impede progress. Miscommunication or misalignments can turn simple tasks into bureaucratic nightmares. Furthermore, these issues can introduce overwhelming and unnecessary complexity, causing missed opportunities. Here are specific examples of these tensions:

#### Technology vs. Organization:

- **Example:** An organization adopts a monolithic IT system that centralizes all operations, making management efficient. However, development teams are forced to navigate bottlenecks when making changes because all teams are intertwined with a single codebase. This centralization results in a traffic jam of dependencies, slowing down deployments and requiring teams to coordinate their work in overly complex ways.
- **Impact:** This structure hinders team autonomy, making rapid iteration and innovation impossible. The teams can't work in isolation, and a slight change in one area can inadvertently affect another, leading to downtime and delays.

#### Product vs. Technology:

- **Example:** An organization has adopted microservices architecture, where each team owns a service domain, allowing for independent development and deployment. However, the product's feature requirements are organized differently—for example, by user experience across multiple domains. As a result, a seemingly simple product change—like a new user interface—requires touching various microservices.
- **Impact:** This creates a situation where cross-team coordination is mandatory for small product changes, akin to playing “Whac-A-Mole.” Every minor modification may ripple across several teams, increasing complexity and slowing progress.

### **Business vs. Product:**

- **Example:** A company’s business strategy frequently shifts between cost-cutting and innovation. At one point, the priority is to reduce operational expenses and hold off on new features. A few months later, the focus shifted toward rapid product innovation and cloud migration to stay competitive. The product teams, meanwhile, are struggling to align their development roadmaps with these conflicting objectives.
- **Impact:** This results in conflicting priorities across teams. While the business insists on reducing costs, the product teams are asked to deliver new features and pivot technology platforms, leading to overworked teams and unsustainable project timelines.

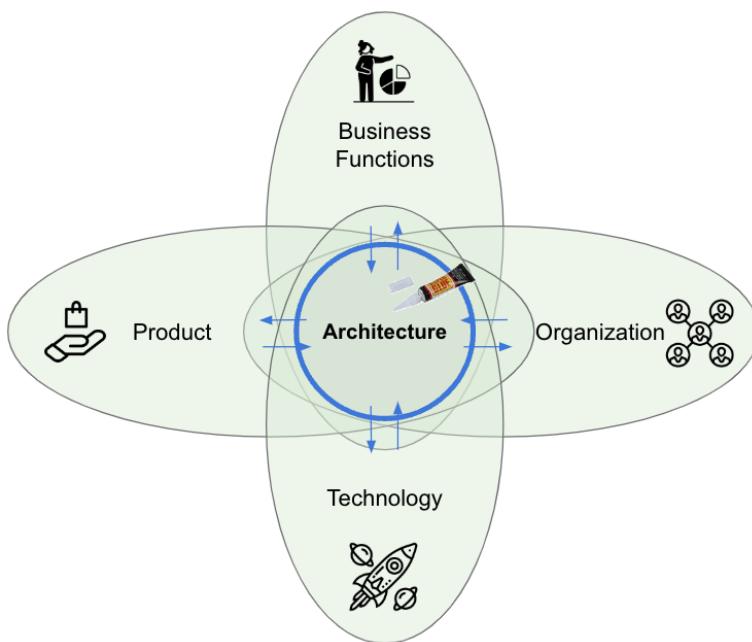
### **Organization vs. Business Functions:**

- **Example:** The organizational structure is designed around siloed departments, such as IT, marketing, and finance, each with different goals and KPIs. However, the business function requires cross-department collaboration to achieve strategic objectives, such as launching a new product or migrating to the cloud. The departments, with conflicting processes and isolated goals, end up miscommunicating and delaying delivery.
- **Impact:** This tension results in project bottlenecks, unclear decision-making, and slow response times as each department operates independently instead of moving toward a shared business objective. The lack of collaboration increases friction and risks missed market opportunities.

When not addressed, these tensions can severely limit an organization's ability to stay competitive, agile, and responsive to market and technological changes.

### 15.2.2: Superglue Role in Reducing Tensions

Superglue architects should be able to navigate the wild complexities of modern organizations. They should facilitate that the various architectures—business, product, technology, or organizational design—don't just coexist but thrive together. Architects' **holistic approach** not only reduces tension and misalignment but reassures the organization's smooth operation, like butter on hot toast.



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

Acting as superglue, the architects can effectively **reduce tensions** among technology, product, organization, and business functions. This act

includes facilitating **critical conversations** between these units and ensuring alignment and cohesion. As depicted in Figure 3, the goal of architecture is not to add new constructs between these four elements, but to **bring them closer together**. By fostering closer relationships and better communication, architects can help these elements work in unison rather than at cross purposes. Here are examples of how architects reduce tensions and ensure these different domains work cohesively:

#### Aligning Technology with Business Goals:

- **Example:** An organization is undergoing a digital transformation, adopting cloud infrastructure to reduce costs and increase scalability. The business wants to shift to cloud services to improve profitability quickly. The architect's role here is to assess the current technology stack and highlight the potential trade-offs between speed, cost savings, and technical debt.
- **Architect's Role:** The architect ensures that the cloud migration strategy aligns with both business cost-cutting goals and the long-term sustainability of the technology stack. By communicating how aggressive cloud migration might lead to technical debt and future maintenance costs, the architect keeps both business and technology stakeholders aligned, avoiding unrealistic expectations or missed deadlines.

#### Balancing Product Roadmaps with Organizational Capabilities:

- **Example:** A product team wants to roll out new features based on customer feedback. However, the development teams are already stretched thin with existing workloads, and the current organizational design doesn't support the speed required for this new product push.
- **Architect's Role:** The architect engages the product team and the organizational leadership to ensure the product roadmap is realistic, given the current team structure and workload. They suggest restructuring teams or adopting different DevOps practices to improve delivery speed. By facilitating these conversations, the architect helps product teams avoid bottlenecks and missed deadlines while aligning with the organization's resource capabilities.

#### Translating Technical Jargon for Business Stakeholders:

- **Example:** The business unit is planning an aggressive cost-cutting strategy to reduce IT expenditures by slashing infrastructure budgets. At the same time, the tech team is preparing for a major system upgrade to improve long-term scalability and security.
- **Architect's Role:** The architect steps in to explain the technical ramifications of cost-cutting to business leaders in non-technical language, such as the risks of underfunding system upgrades or deferring maintenance. They can frame technical debt and potential security vulnerabilities in terms of business risk, making it easier for business leaders to see the bigger picture. This understanding can ensure that critical upgrades are not sacrificed for short-term savings.

### Ensuring Product and Technology Co-Evolution:

- **Example:** The technology team wants to adopt a new containerized microservices architecture to improve scalability, while the product team is still focused on rolling out features built on an outdated monolithic system. The mismatch creates tension between rapid product releases and long-term technology goals.
- **Architect's Role:** The architect can ensure that both teams' goals are aligned by creating a phased migration plan. They can propose that the product team start developing new microservices architecture features while maintaining the monolithic system until a complete transition is feasible. This dual track can enable continuous feature delivery without sacrificing long-term technical goals.

### Facilitating Critical Conversations Across Teams:

- **Example:** An organization's technology team pushes for agile methodologies to enable rapid development and deployment. At the same time, the business unit still operates on annual planning cycles, making it challenging to sync timelines.
- **Architect's Role:** The architect can organize cross-team workshops to help both sides understand each other's processes and goals. By explaining how agile practices can enhance product delivery speed and adjust to market needs faster, they can help the business side adjust its planning cycles. This explanation can reduce the friction between long-term planning and agile execution, fostering better alignment between business and technology functions.

### Mitigating Conflicting Business Objectives:

- **Example:** The business leadership wants to reduce costs and increase product innovation simultaneously, which may place contradictory demands on the IT department. Innovation requires investment in new technologies and R&D, while cost-cutting restricts resources.
- **Architect's Role:** The architect can propose solutions that balance both goals, such as adopting cloud services that can reduce operational costs while offering flexibility for innovation. By integrating the right cloud-native solutions, the architect can ensure the company can innovate without sacrificing budget constraints, thus harmonizing conflicting business objectives.

The “super glue architect” helps navigate these tensions by being deeply involved in the organizational, technical, and product conversations. They bridge the gap between different stakeholders, ensuring smooth collaboration. Their success is measured by the seamless operation of business, product, and technical goals, even when their role becomes less visible over time.

## 15.3: Supergluing In Action #2: Aligning Discussions About Problems, Solutions and Implementations

The [Theory of Constraints \(TOC\)](#)<sup>4</sup>, developed by Eliyahu M. Goldratt, is another source of inspiration I used to work as a superglue architect. TOC is a management philosophy focusing on identifying and addressing the most critical bottleneck (constraint) to improve overall performance.

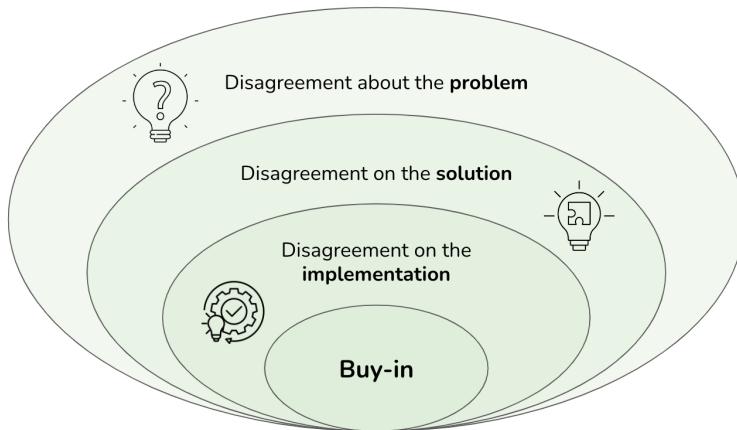


image by peopleimages from istock

While Goldratt's work is much broader, the part I find inspiring is his view on resistance to change, which he sees as manifesting in three forms: disagreement about the problem, disagreement on the solution, and disagreement on the implementation (Figure 4). This model is a useful guide to help people resolve disagreements. This model can also be useful in showing that many disagreements stem from people not realizing they are talking about different topics (e.g., trying to decide on low-level technology choices while there is no agreement on which problem they are solving).

---

<sup>4</sup>[https://en.wikipedia.org/wiki/Thinking\\_processes\\_\(theory\\_of\\_constraints\)](https://en.wikipedia.org/wiki/Thinking_processes_(theory_of_constraints))



**Figure 4:** Goldratt's view on resistance to change: disagreement about the problem, disagreement on the solution, and disagreement on the implementation.

### 15.3.1: Disagreement about the Problem (What to change?)

Disagreement about the problem occurs when stakeholders or team members perceive the issue differently. In IT architecture and software engineering, this can lead to significant delays and inefficiencies because you might direct resources toward solving a problem that isn't the root cause of the constraint.

For example, consider an IT team responsible for maintaining an e-commerce platform. The platform is experiencing slow response times, as users have reported. The database team believes the issue lies in inefficient database queries, while the network team thinks the problem is due to network latency. Meanwhile, the backend application developers suspect poorly optimized code. UX designers think that reported slowness is happening on the front end and is a consequence of the poor implementation of the user interface. These teams cannot agree on the primary problem. They may waste time optimizing areas that aren't the real bottleneck, leading to a patchwork of partial solutions that don't fully resolve the performance issue.

### **15.3.2: Disagreement on the Solution (What to change into?)**

Once you identify a problem, the next challenge is agreeing on the best solution. This disagreement can stem from differing experiences, knowledge, or biases toward particular technologies or methodologies.

In the same e-commerce platform example, the team might disagree on the solution after identifying the database as the root cause of the problem. The database administrators might suggest migrating to a more powerful database server, while the developers propose rewriting the queries more efficiently. UI developers think they should significantly refactor frontend code to a new version of the UI library they wanted to use before but did not have time for the work. The operations team might push for a backend redesign and to scale the entire system horizontally by adding more servers. These conflicting solutions can cause delays as the team debates the best approach, potentially leading to a suboptimal compromise or a solution that introduces new problems.

### **15.3.3: Disagreement on the Implementation Approach (How to cause the change?)**

The implementation can still be contentious even when you agree upon a solution. The implementation phase is critical, as poor execution can negate the benefits of the chosen solution.

Continuing with the e-commerce platform scenario, suppose the team agrees to optimize the database queries as the solution. Disagreements might arise regarding how to implement these changes. The development team might want to rewrite all queries in one go, while the operations team might prefer a phased approach to minimize downtime. Additionally, there might be disagreements about the testing environment, deployment strategies, or even the timeline. If you do not resolve these disagreements, the implementation could be delayed or poorly implemented, leading to new issues such as data integrity problems or complete system outages.

### **15.3.4: Confusion about the Level of Disagreement**

A more difficult-to-spot type of confusion can arise when people are unclear about the level of disagreement they are addressing. This misalignment can be particularly problematic in IT architecture and software engineering, where project complexity often involves multiple layers of decision-making.

People may think they are discussing and disagreeing on one aspect of a project—such as the implementation—when their disagreement stems from a deeper issue, such as not being aligned on the problem itself. This understanding can lead to more productive discussions, efficient use of effort, and decisions that effectively address the core issues.

#### **15.3.4.1: Example 1: Misalignment on the Problem vs. Implementation**

Imagine a software development team optimizing a web application's performance. The team starts debating how to implement changes—whether to refactor code, adjust server configurations, or optimize the database. However, underlying this discussion is a fundamental misalignment: they disagree on the problem. Some team members believe the issue is with the application code, while others think it's the database or the server infrastructure, and third think the problem is poor UX design.

As they discuss implementation strategies, they may argue whether to prioritize refactoring code or upgrading hardware, going into a heated debate about low-level technology choices and versions of the libraries they plan to use. But this debate is fruitless because they haven't first aligned on the problem. The result is that they might implement a solution that doesn't effectively address the real issue, or they might go in circles without reaching a consensus, leading to delays and frustration.

#### **15.3.4.2: Example 2: Misalignment on the Solution vs. Problem**

Consider a scenario where an IT department is trying to improve the security of its systems. The team is debating whether to implement a new firewall or upgrade its encryption protocols. They might think they disagree on the solution—firewall vs. encryption—but the real issue is that they haven't agreed on the specific security problem they are trying to solve.

Some team members might be focused on external threats, while others are more concerned about internal data breaches. Because they are not aligned on the problem, their discussion about the solution is confused. One group pushes for a solution that addresses external threats, while another advocates for measures that protect against internal risks. Without recognizing this misalignment, the team risks implementing a solution that only partially addresses the organization's security needs.

### 15.3.5: Superglue Role in Resolving Disagreements

As connective tissue, the superglue architect can help resolve disagreements with effective communication, clear documentation, and a collaborative approach to the problem:

1. **Clearly Identify and Agree on the Problem:** Before discussing solutions or implementation, ensure that everyone has a shared understanding of the problem.
2. **Differentiate Between Problem, Solution, and Implementation:** Facilitate team members to recognize and articulate their discussion level. This differentiation helps prevent confusion and keeps discussions productive.
3. **Facilitate Clear Communication:** Use structured methods like root cause analysis or problem statements to align everyone on the problem before discussing potential solutions and implementations.
4. **Seeing Multiple Dimensions:** As [Gregor Hophe elaborated nicely<sup>5</sup>](#), when architects encounter stalemate situations, they may try to find a new model to demonstrate that everyone is simply looking at the same thing from different perspectives.

By being aware of the potential for confusion and actively working to clarify the level of disagreement, teams can make more informed decisions, implement effective solutions, and, ultimately, improve project outcomes.

---

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

## 15.4: Superglue Impact: Keeping Everyone in the Same Boat, Upon a Stormy Sea

While architects stay close to the technology, they must ensure it works for the business and customers, not the other way around. By keeping everyone **in the loop** and **aligned**, architects help clear the many **pitfalls of misalignment** and keep the organizational machine running smoothly. Their role is to facilitate communication, ensure alignment, and guide the organization toward cohesive and practical solutions, preventing the myriad risks associated with misalignment.



image by mbbirdy from istock

Misalignment between these elements can introduce several **key risks**, which superglue architects need to be aware of and can mitigate:

- **Building the Wrong Products:** If technology implementation is based on incorrect assumptions, you might create a product that doesn't meet the actual people needs. Misalignment can lead to producing irrelevant solutions, like trying to sell snow boots in the Sahara.
- **Wrong Prioritization of Activities:** Without clear business and product metrics, resources might be directed towards developing

“interesting” but non-valuable products. Proper alignment ensures development efforts are focused on initiatives that add real value, not on quirky side projects.

- **Unexpected Delivery Delays:** Misalignment can lead to underestimating projects’ complexity, effort, and dependencies, causing significant delays. This misalignment can make a project feel perpetually stuck, like being in a time loop where unforeseen obstacles continuously hinder progress.
- **Duplication of Effort:** Without harmonization across business and product strategies, efforts may be duplicated, leading to inefficiencies. This duplication is akin to repeatedly reinventing the wheel, which is wasteful and counterproductive.
- **Building Too Complex Products:** Overly complex and configurable systems can be developed to address all possible scenarios, resulting in cumbersome solutions where simpler, more harmonized approaches would suffice. This complexity is like using a Swiss Army knife when only a spoon is needed—overkill and overly complicated.
- **Overengineering:** Without pushback to simplify products and a lack of understanding of the technology, overengineering can occur. Imagine using a monster truck for a grocery run—impressive, but entirely unnecessary.
- **Building Too Simple, Unscalable Products:** Assuming processes will be simple when, in reality, essential complexity needs to be supported, can lead to fragile and rigid systems.
- **Building Low-Quality Products:** Unnecessary complexity and lack of critical knowledge and expertise can lead to low-quality products that fail under pressure, like a dollar-store umbrella in a hurricane. Ensuring quality requires aligning expertise and simplifying designs where possible.
- **Complicated Dependencies Between Teams:** Suboptimal organizational design and lack of awareness of system and team dependencies can slow down coordination, creating a bureaucratic nightmare. Efficient team structures and clear communication channels are crucial to maintaining momentum.
- **Creating Fragile, Unsustainable Team Structures:** Relying on a small number of developers for critical technologies can make teams extremely vulnerable. Building resilient team structures with adequate support and redundancy is essential.

## 15.5: To Probe Further

- Thinking Like an Architect<sup>6</sup>, by Gregor Hohpe, 2024
- Architects See More Dimensions<sup>7</sup>, by Gregor Hohpe, 2020
- Architects Zoom In and Out, See Different Things<sup>8</sup>, by Gregor Hohpe, 2020
- Architects Look For Causality<sup>9</sup>, by Gregor Hohpe, 2020
- Architects See Shades of Gray, Look for Balance<sup>10</sup>, by Gregor Hohpe, 2020
- Here's why enterprise IT is so complex<sup>11</sup>, by Gregor Hohpe, 2018

---

<sup>6</sup><https://www.infoq.com/articles/thinking-like-architect/>

<sup>7</sup><https://architectelevator.com/architecture/multiple-dimensions/>

<sup>8</sup><https://architectelevator.com/architecture/architects-zoom/>

<sup>9</sup><https://architectelevator.com/architecture/architects-causality/>

<sup>10</sup><https://architectelevator.com/architecture/architects-shades-of-gray/>

<sup>11</sup><https://architectelevator.com/architecture/it-complexity/>

## 15.6: Questions to Consider

Being a superglue architect means constantly developing and redefining 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?*

# **16: Thinking Like an Architect: Balancing Curiosity, Doubt, Vision, and Skepticism**



image by vawiley from istock

**IN THIS SECTION, YOU WILL:** Understand that balancing curiosity, doubt, vision, and skepticism is essential for driving sustainable innovation and change in organizations.

#### KEY POINTS:

- **Curiosity and wonder** spark exploration, leading to technological breakthroughs. However, without caution, it can result in premature adoption of immature solutions.
- **Doubt** forces a critical evaluation of progress, ensuring that innovative ideas are grounded in practical and validated approaches. Over-reliance on doubt can stifle risk-taking and hinder breakthrough innovation.
- **Vision and belief** provide a long-term perspective, guiding efforts toward significant, transformative goals. Yet unchecked vision can lead to confirmation bias or misdirected resources.
- **Skepticism** helps prevent overcommitment to unproven ideas, ensuring teams remain grounded. However, excessive skepticism can result in missed opportunities and discourage creative risk-taking.

- Architects must constantly reflect on how well they balance these forces. In doing so, they can ensure that their efforts are driven by a healthy combination of exploration, critical validation, strategic guidance, and cautious realism—all crucial to achieving lasting success in any organizational change.

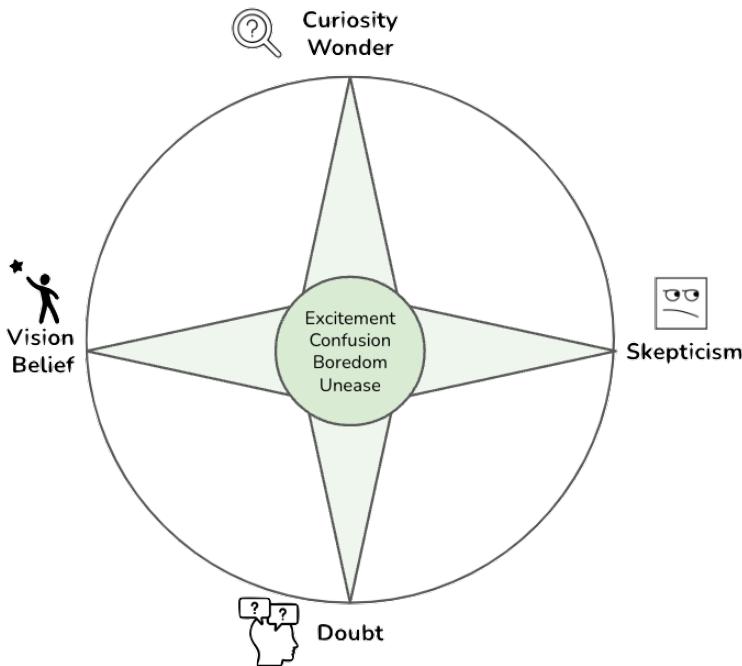
IT architects must help organizations innovate and transform responsibly and sustainably. They must bring new ideas and support or voice concerns about others' plans and proposals here. In pursuing **sustainable innovation** and **organizational change**, understanding the underlying motivators that drive individuals and teams is essential. Drawing on my previous research, I propose the [metaphor of a compass](#)<sup>1</sup> to illustrate how balancing four fundamental forces—curiosity, doubt, vision, and skepticism—guides this process. Each of these motivators plays a distinct but interconnected role (Figure 1):

- **Curiosity** sparks exploration,

---

<sup>1</sup><https://researcher-practitioner.com/research-compass>

- **Doubt** ensures deeper scrutiny,
- **Vision** aligns efforts with long-term goals, and
- **Skepticism** questions feasibility and assumptions.



**Figure 1:** The compass for driving sustainable innovation and change in organizations.

IT architects can be crucial in recognizing these different motivators and actively supporting contributions from all four. Their role is to ensure that:

- **Curiosity** is harnessed productively through experimentation and exploration.
- **Doubt** and **skepticism** are encouraged to critically evaluate ideas without stifling innovation.
- **Vision** aligns the work with strategic priorities, ensuring that technical innovations contribute to broader organizational goals.

The question for organizations and IT architects is not about choosing one direction (whether curiosity, vision, etc.) but about finding the delicate **balance between all four forces**. This balance drives sustainable innovation, allowing for creative growth tempered with critical thought and long-term planning.

By balancing these motivators, organizations can successfully navigate the **complexities of innovation**, ensuring that change is not only driven by bold ideas but also thoroughly validated, strategically aligned, and pragmatically grounded.

In the following sections, I describe these four forces in more detail.

## 16.1: Curiosity and Wonder

Curiosity and wonder are often the most intuitive and powerful motivators for change. Curiosity usually drives innovation—developers and engineers are drawn to explore new technologies, tools, or methods, and their **natural desire to learn** fuels progress. Organizations that encourage autonomy and innovation create environments where curiosity thrives.

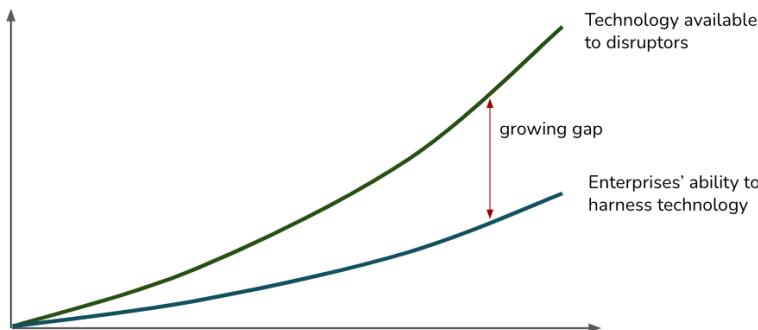


image by mtstock studio from istock

Architects too frequently become less curious and focus on reviewing and criticizing other ideas. However, I see them as critical drivers for the **responsible exploration of new technologies**. Architects have a background knowledge that enables them to quickly understand the pros and cons of new directions. As per [ThoughtWorks analysis<sup>2</sup>](#), there is a massive gap between the exponential growth of technology and organizations' ability to harness it, and this gap will only increase with time. Staying curious and being able to follow new technology developments is a crucial competitive advantage of any organization.

---

<sup>2</sup><https://www.thoughtworks.com/en-ec/insights/blog/seven-tenets-new-age-business-analysis>



**Figure 2:** Over time, technology grows at an exponential rate. But businesses aren't able to keep up fast enough. Curiosity is an important force that helps organizations adopt new technologies. Source: thoughtworks.com

However, over-reliance on curiosity can lead to a focus on novelty rather than sustainability. Rapid, uncritical adoption of new technologies may lead to **naive solutions**, overlooking potential risks or **technical debt**. There is a fine line between innovation fueled by curiosity and a risk of unsustainable growth if curiosity is not tempered with critical thinking.

Here are a few examples where curiosity and wonder have driven innovation yet also highlight the risks of uncritical adoption of new technologies:

- **Microservices Architecture**
  - *Curiosity and innovation:* The shift from monolithic architectures to microservices was driven by curiosity and the need to solve scalability and flexibility issues. Developers were fascinated by decoupling services to allow for independent deployment and scaling, which led to significant innovations in cloud-native development, such as containerization and Kubernetes.
  - *Risk of over-reliance on novelty:* However, organizations that rushed to adopt microservices without understanding the complexity of managing distributed systems faced orchestration, monitoring, and increased operational overhead issues. In some cases, the shift introduced technical debt, making systems more complex to maintain without proper tooling.

- **Serverless Computing**

- *Curiosity and innovation:* Serverless architectures emerged due to developers' curiosity about reducing infrastructure management. The desire to focus solely on writing business logic and letting cloud providers handle scaling and infrastructure led to major efficiency gains, especially for dynamic workloads.
- *Risk of uncritical adoption:* On the downside, developers who rushed into serverless without considering factors like cold start delays, vendor lock-in, or limited control over execution environments ended up with architectures that were costly or hard to maintain in the long term.

- **Agile Methodology**

- *Curiosity and innovation:* Agile methodologies like Scrum and Kanban grew from a desire to overcome the rigidity of traditional waterfall project management. The idea of iterative development cycles and continuous improvement resonated with engineers who wanted more flexibility in their work.
- *Risk of novelty over sustainability:* Some organizations adopted Agile in name only, leading to chaotic workflows without truly understanding the principles of collaboration and continuous feedback. This has sometimes resulted in “fragile Agile” environments, where the novelty of faster iterations creates pressure without improving actual delivery outcomes.

- **Blockchain in Software Development**

- *Curiosity and innovation:* Blockchain's promise of decentralized and immutable records intrigued many in software engineering, driving curiosity to explore use cases beyond cryptocurrency, such as secure voting systems, supply chain tracking, and data integrity.
- *Risk of uncritical adoption:* However, in many instances, blockchain was applied to problems that did not require such a complex solution, leading to increased costs, slower performance, and added technical debt when more straightforward databases would have sufficed.

Curiosity has driven **innovation and progress** in all of these examples, but **unchecked enthusiasm** for new technologies can lead to long-term challenges if sustainability and critical evaluation are overlooked.

## 16.2: Doubt

When we make some visible progress, we may ask ourselves if our results are **wrong**, **coincidental**, or a result of **wishful thinking**. Such questions are the beginning of doubt, one of the most important motivators behind creating robust results. Any evaluation can be viewed as an effort to reduce doubts about our findings.

Doubt is a critical motivator that drives rigorous validation and testing of new ideas. Doubt is primarily a **positive force**. Contrary to skepticism, doubt does not question the possibility of knowing something or the validity of pursuing some direction. When we doubt some finding, we want to set it on **firmer ground** and add **more certainty**.



image by hiraman from istock

In software engineering, doubt manifests in practices like code reviews, testing frameworks, and peer reviews of architectural designs. IT architects must foster a **culture of constructive doubt**, where teams constantly question whether their solutions are robust, scalable, and fit for purpose.

However, excessive doubt can **stifle innovation**. If teams focus only on incremental improvements and avoid risk, they may miss opportunities for breakthrough innovations. Balancing doubt and innovation is essential to creating sustainable, high-performing solutions.

Here are a few examples from IT architecture and software engineering where doubt drives rigorous validation, ensuring the robustness of systems while balancing innovation and risk:

- **Unit Testing and Test-Driven Development (TDD)**

- *Doubt and validation:* In software engineering, doubt about whether code functions (and will keep functioning) correctly leads to the widespread use of unit testing and Test-Driven Development (TDD). Engineers write tests to ensure that each piece behaves as expected. This rigorous approach minimizes errors, reduces technical debt, and ensures robust software.
- *Excessive doubt risk:* However, focusing too much on testing and validation can slow development. In some cases, teams may overly rely on TDD, creating rigid tests that prevent refactoring and limit innovative solutions, stifling the exploration of new design patterns.

- **Code Reviews**

- *Doubt and validation:* Code reviews are driven by doubt that any single developer's solution is flawless. Engineers critically examine each other's code to identify flaws, improve code quality, and enforce consistency. This process helps reduce bugs, ensures compliance with design principles, and fosters knowledge sharing.
- *Excessive doubt risk:* If the review process becomes overly critical, it can lead to delays, a decrease in team morale, and resistance to experimenting with novel approaches. An overly cautious environment may discourage developers from innovating.

- **Continuous Integration/Continuous Deployment (CI/CD) Pipelines**

- *Doubt and validation:* CI/CD pipelines are rooted in doubt about whether the software will behave correctly in different environments. Automated tests, builds, and deployments are triggered to ensure that changes don't break functionality, and every small change is validated before release. This approach minimizes errors and makes systems more robust.

- *Excessive doubt risk:* An over-reliance on automation can sometimes create rigid workflows. Teams may hesitate to make significant architectural changes or take on ambitious projects, fearing that the automated checks will flag too many issues or create bottlenecks.

- **Architectural Peer Reviews**

- *Doubt and validation:* In IT architecture, peer reviews of system designs are based on the doubt that a single architect's vision is complete. These reviews focus on scalability, security, and maintainability, ensuring that systems are designed to meet long-term goals and handle future demands.
- *Excessive doubt risk:* Excessive critique in these reviews can lead to overly conservative decisions. Architects may avoid riskier, innovative technologies that could offer better performance or scalability out of fear that they introduce too much uncertainty or complexity.

- **Security Testing (Penetration Testing and Threat Modeling)**

- *Doubt and validation:* Security is an area where doubt is especially critical. Practices like penetration testing and threat modeling are based on the assumption that systems are vulnerable. Engineers actively seek out system weaknesses to minimize risks and improve overall security posture.
- *Excessive doubt risk:* If security concerns are over-prioritized, they can block new features, slow development, and create a climate where innovation is too risky. This balance is particularly tricky in industries like fintech and healthcare, where innovation and security must coexist.

In all these cases, doubt catalyzes the creation of **robust, validated solutions**. Yet, fostering a balance between doubt and innovation is essential to ensure teams don't shy away from bold new ideas.

## 16.3: Vision and Belief

While the term belief may have a negative connotation, it is difficult to imagine any transformation activity without some form of belief or guiding vision. Vision and belief are **long-term strategic motivators** that guide large-scale change initiatives. Successful IT architecture often begins with a bold vision—such as a belief in the transformative potential of a new technology.



image by georgepeters from istock

However, vision without critical evaluation can lead to **confirmation bias**. Architects must balance visionary thinking with the rigor of curiosity and doubt to avoid pursuing misguided or overly narrow goals.

Vision and belief are much more complex motivators than curiosity and wonder. When driven by curiosity, we follow interests and the desire to learn something new. Vision and belief, on the other hand, require a **longer-term commitment** to some idea and constant effort to focus and organize our activities. While curiosity and wonder encourage exploration, vision and belief guide and sustain efforts toward transformative outcomes.

Here are some examples where vision and belief played critical roles in driving large-scale change initiatives and transformative innovations:

- The Internet

- *Vision and belief:* Early pioneers like Vannevar Bush and J.C.R. Licklider believed in a global network connecting people, information, and machines. Their vision of an “[Intergalactic Computer Network](#)”<sup>3</sup> became the foundation of the modern internet. Licklider’s belief in the collaborative potential of interconnected computers drove funding and research, eventually leading to the ARPANET, the precursor to today’s internet.
- *Balancing with critical evaluation:* Licklider’s vision was tempered by ongoing research and testing to solve problems like network protocols, security, and scalability. Engineers balanced this visionary belief with the practical work of building resilient, fault-tolerant systems.

- Agile Development

- *Vision and belief:* The Agile Manifesto was born from the belief that traditional waterfall software development was too rigid and inefficient. Visionary software engineers believed in a more collaborative, iterative approach that would allow for faster delivery, continuous feedback, and greater adaptability. This belief led to the widespread adoption of Agile methodologies, revolutionizing how software is developed and deployed.
- *Balancing with critical evaluation:* While Agile was visionary, its success depended on carefully testing its principles in real-world projects. Teams needed to critically evaluate which Agile practices worked best for them. Excessive belief in Agile without adaptation has sometimes led to failed implementations, as teams focused too much on process over outcomes.

- DevOps Movement

- *Vision and belief:* The DevOps movement was driven by a belief in the transformative potential of breaking down silos between development and operations teams. Early DevOps pioneers believed collaboration, automation, and continuous delivery could improve software quality and speed dramatically. This vision transformed how organizations manage software lifecycles, driving innovation in automation tools like Jenkins, Docker, and Kubernetes.

---

<sup>3</sup>[https://en.wikipedia.org/wiki/Intergalactic\\_Computer\\_Network](https://en.wikipedia.org/wiki/Intergalactic_Computer_Network)

– *Balancing with critical evaluation:* While DevOps was visionary, teams had to critically evaluate its feasibility in different contexts. Not every company could seamlessly adopt a DevOps culture, and early adopters had to address the challenges of tooling, integration, and operational complexity to ensure long-term success.

- Artificial Intelligence (AI) and Machine Learning (ML)

– *Vision and belief:* Visionaries like Alan Turing and, later, engineers at companies like Microsoft and Google believed in the transformative potential of AI and machine learning. Their belief in creating machines that could learn and think has led to breakthroughs like natural language processing, autonomous systems, and AI-driven decision-making. This vision continues to push the boundaries of computing and software engineering.

– *Balancing with critical evaluation:* The AI and ML fields are filled with successes and failures. While belief in AI's potential is substantial, developing robust AI systems requires extensive testing, validation, and doubt about these technologies' reliability, ethics, and real-world application. Without a balance of critical evaluation, belief in AI's potential could lead to inflated expectations and misuse.

In all these examples, vision and belief served as powerful, **long-term forces** that propelled transformative technological changes. However, these bold ideas were successful because they were accompanied by curiosity, rigorous doubt, and constant critical evaluation to ensure their feasibility and sustainability in the real world. These revolutionary ideas might have failed to materialize without balancing belief with practical testing and adaptation.

## 16.4: Skepticism

Skepticism is a loaded term with several definitions. Closest to the meaning I use here is the Wikipedia definition of skepticism as “*doubt regarding claims that are taken for granted elsewhere*”. Similarly, I view skepticism as a **reality checker** that questions the fundamental premises we usually take as a given. As such, skepticism can call attention to the **viability, feasibility, or practicality** of a direction or approach. Contrary to doubt, which can motivate us to investigate some topic further to obtain more certainty, skepticism may call us to abandon some lines of inquiry and consider alternatives.



image by izusek from istock

Fred Brooks's paper “**No Silver Bullet—Essence and Accidents of Software Engineering**”<sup>4</sup> is probably one of the best examples of helpful skeptical thought in software engineering. Brooks expressed his skepticism toward approaches to software engineering research that aim to discover a single solution that can improve software productivity by an order of magnitude. Brooks seriously questioned the possibility of ever finding such “startling breakthroughs,” arguing that such solutions may be inconsistent with the nature of software. Brooks also made clear that his **skepticism**

---

<sup>4</sup>[https://en.wikipedia.org/wiki/No\\_Silver\\_Bullet](https://en.wikipedia.org/wiki/No_Silver_Bullet)

**is not pessimism.** While Brooks questioned the possibility of finding a single startling breakthrough that would improve software productivity by an order of magnitude, he believed that such improvement can be achieved through disciplined, consistent effort to develop, propagate, and exploit a number of smaller, more modest innovations.

Skepticism plays a crucial role in keeping organizations grounded. In IT architecture, skepticism questions whether specific approaches or technologies are viable. Skeptical thinking can help **prevent costly mistakes** by challenging assumptions and forcing teams to consider alternative approaches. This type of critical thinking is essential for ensuring that decisions are sound and that innovation is pursued responsibly.

Yet too much skepticism can lead to **missed opportunities**. Overly critical attitudes can reject promising innovations before they've had a chance to prove themselves. Furthermore, if skepticism is not a well-argued result of long experience, it may trigger an emotional debate without contributing much to it. Similar to vision, practical skepticism requires deep knowledge and a fundamental understanding of the field.

Here are a few examples where skepticism has played a crucial role in questioning underlying assumptions, helping to prevent misguided decisions while still allowing space for innovation:

- **Monolithic vs. Microservices Architecture**
  - *Skepticism about microservices:* While many organizations have embraced microservices as the modern solution for building scalable and flexible systems, some IT architects have expressed skepticism about whether microservices are the right solution for every use case. This skepticism arises from concerns over complexity, the operational overhead of managing numerous services, and potential performance bottlenecks.
  - *Impact:* Skeptical architects have questioned the assumption that breaking down every application into microservices is inherently better, advocating for alternatives such as modular monolithic architectures. This skepticism has led some teams to avoid unnecessary complexity and better tailor their architectural choices to the project's needs.
- **Blockchain for Everything**

- *Skepticism about blockchain use cases:* Blockchain technology has been widely hyped as a transformative innovation for everything from finance to supply chains to healthcare. However, skeptics have questioned whether blockchain is necessary or even practical for many of these use cases, citing concerns about performance, scalability, and the complexity of decentralized systems.
- *Impact:* This skepticism has led to a more nuanced and careful consideration of where blockchain is a good fit versus where traditional databases or systems are more effective. As a result, many organizations have avoided costly mistakes by not adopting blockchain for applications where it offers little to no advantage.

- Artificial Intelligence (AI) Hype

- *Skepticism about AI's current capabilities:* While AI is often viewed as the future of many industries, there is considerable skepticism about its real-world capabilities, particularly in areas like fully autonomous driving, generalized AI, and complex decision-making. Skeptics question whether AI technology, as it currently stands, can live up to the hype without significant advancements in data, algorithms, and ethics.
- *Impact:* This skepticism has led to a more cautious and realistic approach to AI adoption. Organizations are more likely to adopt AI that can offer tangible, practical benefits (e.g., automation of routine tasks, pattern recognition) rather than diving headfirst into ambitious, underdeveloped AI projects.

- Agile Methodology Skepticism

- *Skepticism about Agile's universality:* While Agile methodologies have been highly successful for many teams, some engineers and IT architects remain skeptical about its applicability in all contexts. Critics argue that Agile when applied dogmatically, can lead to chaotic environments where long-term planning and architectural considerations are neglected.
- *Impact:* This skepticism has encouraged organizations to adopt Agile principles to fit their needs rather than blindly adopting the methodology. Some teams blend Agile with

other structured approaches to balance flexibility and long-term planning, leading to more sustainable development practices.

- **Skepticism about “Serverless” Computing**

- *Skepticism about the limits of serverless:* Serverless computing promises to abstract away the complexities of managing infrastructure, allowing developers to focus on writing code. However, skeptics point out limitations such as cold starts, vendor lock-in, and the complexity of debugging in serverless environments.
- *Impact:* This skepticism has led some organizations to avoid over-committing to serverless architectures for mission-critical applications. Instead, they use serverless selectively, adopting hybrid approaches that balance the benefits of serverless with the control offered by traditional infrastructure.

In these examples, skepticism has served as a valuable “reality check,” challenging assumptions that may otherwise be taken for granted. While skepticism can prevent teams from rushing into unproven technologies, it’s also essential to strike a balance—excessive skepticism can stifle innovation and prevent organizations from exploring potentially transformative ideas. Skeptics in IT architecture and software engineering remind us to be thoughtful, deliberate, and pragmatic in approaching change.

Skepticism is probably one of the most complex forces. Contrary to doubt, which can rely on several tools and methods (e.g., experiments, ethnographical methods), there are no simple and structured tools for skepticism. Practical skepticism requires careful thought, experience, and an excellent overview of the field.

## 16.5: Balancing Motivators in IT Architecture

The four points of the compass—curiosity, doubt, vision, and skepticism—are not mutually exclusive. Effective IT architecture teams **blend these motivations** into their work. For example, curiosity may drive the exploration of new technology, while doubt leads to rigorous testing, and vision provides the strategic framework for long-term development.



image by happyphoton from istock

Balancing these motivators at an organizational level is essential for fostering a culture of innovation, resilience, and continuous improvement. Without vision and belief, there would be no forward momentum; without skepticism and doubt, there would be no accountability or critical evaluation.

To guide teams, IT architects should ask the following questions:

- Are we exploring new technologies and approaches with enough curiosity or only looking at things we already know?
- Do we have a clear long-term vision guiding our architectural decisions? Or are we blindly following industry trends, jumping from one hype to another? A well-defined vision not only steers

our actions but also fuels our determination to achieve our long-term goals.

- Are we using appropriate methods to validate our solutions and address doubts?
- Are we skeptical enough of our own work, or are we being overly critical of innovative ideas?

By answering these questions, IT architects can help teams ensure they make **informed, balanced decisions** that sustainably drive organizational development. The goal is to create a harmonious environment where change is driven by curiosity, vision, doubt, and skepticism—leading to more sustainable and impactful outcomes.

## 16.6: To Probe Further

- The Four Points of the Research Compass<sup>5</sup>, by Željko Obrenović, 2013

---

<sup>5</sup><https://researcher-practitioner.com/research-compass>

## 16.7: Questions to Consider

- *How does curiosity drive innovation in your organization, and how do you prevent it from leading to unsustainable or risky decisions?*
- *In what ways can doubt contribute to the quality and robustness of your work, and how do you ensure that it doesn't stifle innovation?*
- *How do you balance long-term vision and belief with the need for critical evaluation and validation in your decision-making?*
- *How do you cultivate constructive skepticism without allowing it to turn into cynicism that hampers creativity?*
- *When exploring new technologies, how do you ensure you follow industry trends and align them with a clear strategic vision?*
- *How can fostering a culture of curiosity, doubt, vision, and skepticism lead to more resilient and impactful solutions in your organization?*
- *What strategies can you use to balance risk-taking with rigorous validation in the innovation process?*
- *How does your organization support contributions driven by the four motivators: curiosity, doubt, vision, and skepticism?*
- *In what ways can skepticism be used as a tool for improvement without dismissing promising new ideas too early?*
- *How do you foster environments where curiosity and exploration are encouraged but tempered with thoughtful, critical evaluation?*

# 17: Architects' Career Paths



image by richvintage from istock

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

**KEY POINTS:**

- A strong engineering background is essential for architects to make informed technology decisions and build effective relationships with developer teams.
- Moving from an engineering role to an architecture role involves broadening scope, increasing diversity, and developing strong communication and influencer skills.
- Career tracks can include Senior Architects (broader responsibilities), Principal Architects (specialized focus), and Enterprise Architects (aligning technical strategy with business objectives).
- Architecture roles can lead to tech leadership positions such as Engineering Director or Chief Technology Officer (CTO), leveraging strategic vision, decision-making, and leadership skills.

In any job, a **career path** is a sequence of jobs or roles that a person takes on throughout their professional life. It typically involves **progression** from entry-level positions to higher-level roles with increasing responsibilities and compensation. Career paths can be **linear**, with a clear upward trajectory, or **non-linear**, with lateral moves and changes in direction.

In this section, I elaborate on possible career paths of architects. In the Appendix, you can also find additional [resources for managing, developing, and hiring architects<sup>1</sup>](#).

---

<sup>1</sup>[career-resources](#)

## 17.1: Solid Engineering Background

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 informed technology decisions and build effective relationships with developer teams.

Architects with a strong engineering background have hands-on experience with coding, debugging, and problem-solving, which is crucial for understanding the technical complexities and constraints of any project. This practical knowledge allows them to make **more informed** and **realistic decisions** about technology stacks, architectural patterns, and system design.

When architects have a solid engineering foundation, they can **earn the trust and respect of developer teams** more easily. Developers are more likely to follow the guidance and recommendations of someone who has walked in their shoes and understands their daily challenges. An engineering background equips architects with the technical language and concepts necessary to communicate effectively with developers. This shared language helps bridge the gap between high-level architectural visions and the detailed implementation work carried out by development teams.



image by kobus louw from istock

Engineers are trained to think critically and solve complex problems.

This skill set is invaluable for architects, who must navigate technical challenges, identify potential issues, and devise robust solutions that align with business goals.

While the path to becoming a successful architect is multifaceted, a strong engineering background provides a solid foundation upon which to build the necessary additional skills and knowledge.

## 17.2: Entering Architecture Space

While a strong engineering background is essential, architects must also develop additional skills to succeed in their roles. Stepping from an engineering position to an architecture role requires three significant changes:

1. **Broader Scope:** Architects must look beyond individual components to see the system as a whole, considering the interactions and dependencies between various parts. This holistic view is essential for creating cohesive and scalable solutions that integrate seamlessly with the entire ecosystem of the organization.
2. **Higher Diversity:** The work becomes more varied, encompassing different technologies, processes, and organizational needs. Architects need to be versatile, adapting to various domains and understanding how different technologies and processes can be leveraged to solve complex problems. This diversity also includes interacting with different teams and stakeholders, each with unique perspectives and requirements.
3. **Changing Skills:** Communication and influencer skills become crucial to success, as architects need to articulate their vision and persuade others to follow it. They must effectively communicate complex technical concepts to non-technical stakeholders, build consensus, and drive strategic initiatives. This shift emphasizes the importance of soft skills alongside technical expertise.



image by vm from istock

Architects have multifaceted responsibility that requires them to be not only technically proficient but also adept at **understanding and aligning with business goals** and user requirements.

### 17.3: Career Progression in IT Architecture

An architect's path can take many different directions, and the roles of this path can have many different names.

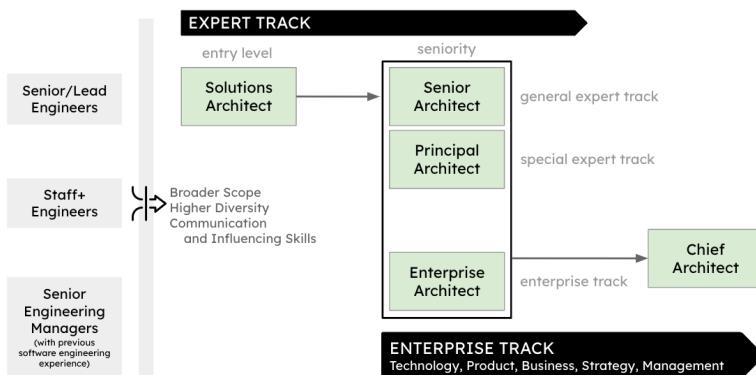


image by bowie15 from istock

People typically enter an IT architecture space as hands-on solution architects. From that position, I usually envision three tracks of progression (Figure 1):

- **Generalist Track (Senior Architects):** These architects are generalists with broader responsibilities who can dig deep into complex issues and identify suitable courses of action. They often navigate from one critical area to another, guided by the organization's direction. Senior Architects play a pivotal role in bridging the gap between different technical teams and ensuring the overall architectural integrity of projects.
- **Specialized Track (Principal Architects):** Senior architects with a specialized track focused on an organization's strategic interest (e.g., data, distributed systems, frontend). Principal Architects dive deep into their specialized areas, providing thought leadership and driving innovation. They often set standards, define best practices, and mentor other architects and engineers.

- **Enterprise Track (Enterprise Architects):** Positioned closer to product, management, strategy, and business functions, Enterprise Architects frequently serve as the right hand to senior engineering leaders. They are responsible for aligning the technical strategy with the business objectives, ensuring that the architecture supports the organization's overall goals. Enterprise Architects often work on cross-functional initiatives and play a key role in strategic decision-making.



*Figure 1: An example of IT architect career paths.*

More important than a formal title is the continuous search for relevance and the ability to **make an impact**. Successful architects are those who continuously learn, adapt, and drive meaningful change within their organizations. They focus on adding value through innovative solutions, effective communication, and strategic alignment with business goals. Whether formally recognized or not, their influence and contributions are pivotal to the success of their teams and the broader organization.

## 17.4: Career Progression Beyond IT Architecture

A career in IT architecture also opens possibilities of pursuing **tech leadership positions** such as Engineering Director or Chief Technology Officer (CTO). The reason for this is multifaceted.



image by miniseries from istock

Firstly, architects develop a strategic vision that **aligns technology with business goals**, a critical skill for any tech leadership role. They gain experience in making high-stakes decisions that impact the entire organization, which is directly relevant to positions like Engineering Director or CTO.

Secondly, architects often **work closely with senior management** and various departments, providing them with a broad perspective on organizational dynamics and strategic planning. This experience is invaluable for leadership roles that require a holistic understanding of both technology and business.

Thirdly, architects' roles require **strong leadership and mentorship abilities**, as they often guide and influence engineering teams. These skills are essential for higher-level leadership positions, where the ability to inspire and manage large teams is crucial.

Lastly, the transition from an architect to a tech leadership role is a natural progression as both roles require a **deep understanding of technology, strategic vision**, and the ability to drive innovation within an organization. This career path leverages the architect's experience in building robust systems and their ability to foresee and mitigate potential risks, ensuring the technology infrastructure supports the organization's long-term goals.

## 17.5: To Probe Further

- Appendix: Resources for Managing, Growing, and Hiring Architects<sup>2</sup>
- Appendix: Architect Archetypes<sup>3</sup>
- Software Architect Archetypes<sup>4</sup>, by Gergely Orosz, 2023

---

<sup>2</sup>[career-resources](#)

<sup>3</sup>[archetypes](#)

<sup>4</sup><https://newsletter.pragmaticengineer.com/p/software-architect-archetypes>

## 17.6: 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: On Human Complexity**

## 18: On Human Complexity: Introduction

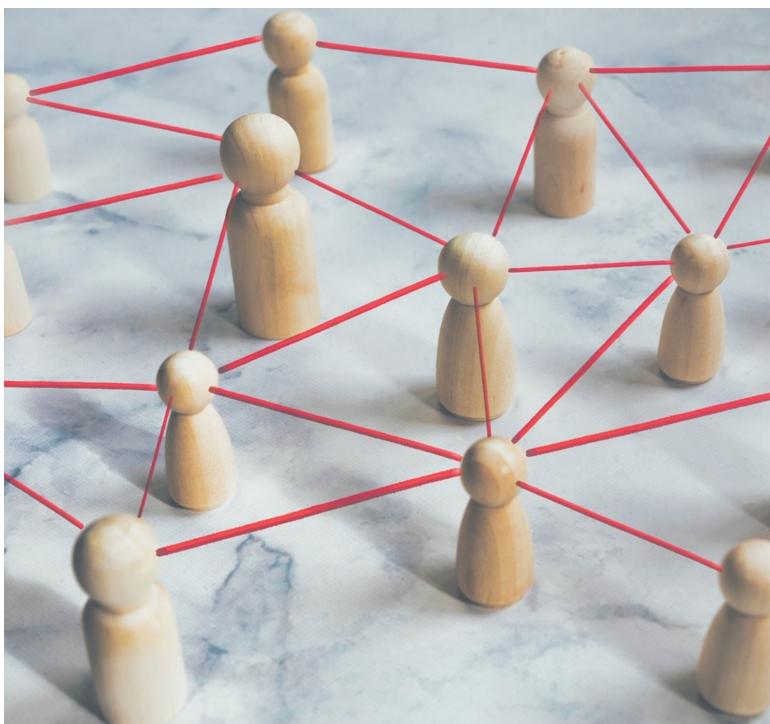


image by cerro photography from istock

**IN THIS SECTION, YOU WILL:** Get a summary of several resources that I use as inspiration for developing awareness of human complexities.

In today's rapidly evolving tech industry, architects face the daunting task of managing complexity while striving for efficiency and innovation. The following sections explore two essential resources that can help architects navigate these challenges and enhance their effectiveness:

The following sections will explore:

- **The Culture Map: Architects' Culture Mindfield Compass:** Navigating cultural diversity is crucial in multinational organizations. Erin Meyer's "The Culture Map" offers invaluable guidance for IT architects to collaborate seamlessly with colleagues from different cultural backgrounds.
- **The Human Side of Decision-Making:** Decision-making is not purely a logical process; it is deeply influenced by human psychology. Architects must be aware of cognitive biases that can skew their judgments. Outcome bias, hindsight bias, and confirmation bias are just a few of the common pitfalls that can affect decision-making. By recognizing these biases, architects can implement strategies to mitigate their effects, leading to more balanced and objective decisions. This understanding is crucial for fostering a culture of rational and evidence-based decision-making within IT teams.
- **Effortless Architecture:** Complexity is a pervasive challenge in the tech industry, often leading to inefficiencies and unsustainable projects. Inspired by Greg McKeown's book "Effortless," this section emphasizes the importance of simplifying tasks and processes. Architects can create more streamlined and effective architectural solutions by focusing on what truly matters and eliminating unnecessary complexity. McKeown's principles encourage architects to adopt a mindset of ease and efficiency, which can lead to more sustainable and manageable projects. Architects can enhance productivity and foster a more innovative environment by reducing the cognitive load and focusing on essential activities.

# 19: 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. **Awareness of cultural differences** is even more important for architects, as they are bridging diverse cultures and domains (technology, business, product, organization).

The work of Erin Meyer, **The Culture Map**<sup>1</sup>, is a beneficial tool for working 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. 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?

---

<sup>1</sup><https://erinnmeyer.com/books/the-culture-map/>

- **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.

## 19.1: Communicating

Architects need to be **good communicators**<sup>2</sup>. But what do we mean when saying someone is a **good communicator**? The responses differ wildly from society to society.



image by luckybusiness from 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).

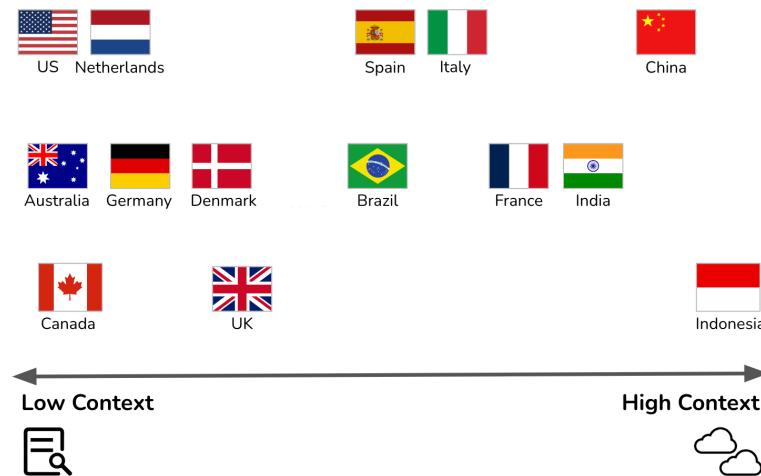
In **low-context** cultures, good communication is **precise, simple, explicit, and clear**. People take messages at face value. Repetition, clarification, and putting messages in writing are appreciated.

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.

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

<sup>2</sup><https://architectelevator.com/strategy/complex-topics-stick/>

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).

In IT architecture and Software Engineering, communication problems often arise due to the diverse cultural backgrounds and professional communities involved in projects. Here are some examples that relate to the dimension of architects needing to be culture-sensitive communicators:

- **Misunderstanding Due to High-Context Communication:**
  - **Scenario:** An architect from a high-context culture presents a new system design to a global team. The architect uses indirect language, implying certain requirements or risks without stating them explicitly, expecting the team to understand the subtleties and read between the lines.
  - **Problem:** Team members from low-context cultures may miss critical nuances or interpret the communication literally,

leading to misunderstandings about the scope, priorities, or potential risks in the project. This issue can result in errors in implementation, overlooked risks, or misaligned expectations.

- **Overwhelming Detail in Low-Context Communication:**

- **Scenario:** An architect from a low-context culture explains a system architecture to a multinational team meticulously, covering every aspect explicitly with extensive documentation.
  - **Problem:** Team members from high-context cultures may find this approach overwhelming, unnecessary, or even insulting, as it could imply a lack of trust in their ability to understand without excessive clarification. This problem can lead to disengagement, frustration, or a failure to consider the system's more implicit or contextual aspects that high-context communicators might naturally consider.

- **Failure to Adapt Communication to Different Professional Cultures:**

- **Scenario:** An architect communicates a technical decision to a non-technical team using complex jargon and technical buzzwords without simplifying or contextualizing the information.
  - **Problem:** Non-technical stakeholders might not understand the implications of the decision, leading to a lack of alignment or support for the project. This problem could result in delays, budget issues, or misinformed strategic decisions that negatively impact the project's success.

- **Cultural Misalignment in Feedback and Collaboration:**

- **Scenario:** During a project review, an architect from a low-context culture directly criticizes a design choice made by a team member from a high-context culture, expecting the feedback to be taken at face value and used for improvement.
  - **Problem:** The team member might perceive the direct criticism as rude or disrespectful, leading to tension or a breakdown in collaboration. In high-context cultures, people often give feedback subtly to maintain harmony, so the direct approach might cause more harm than good.

- **Confusion Over Ambiguous Requirements:**

- **Scenario:** A project involves gathering requirements from stakeholders in a high-context culture, where the stakeholders communicate their needs and expectations implicitly or vaguely.
  - **Problem:** Architects and engineers from low-context cultures might struggle to extract clear, actionable requirements, leading to a design that does not fully meet the stakeholders' expectations. The lack of explicit communication can result in gaps in the final product, requiring rework and causing delays.

These examples highlight the importance of being aware of cultural differences in communication styles and the need for architects to adapt their communication approach based on the context and audience. Balancing between low-context clarity and high-context nuance can help ensure that messages are understood as intended, reducing the risk of miscommunication in complex, multicultural projects.

## 19.2: Evaluating

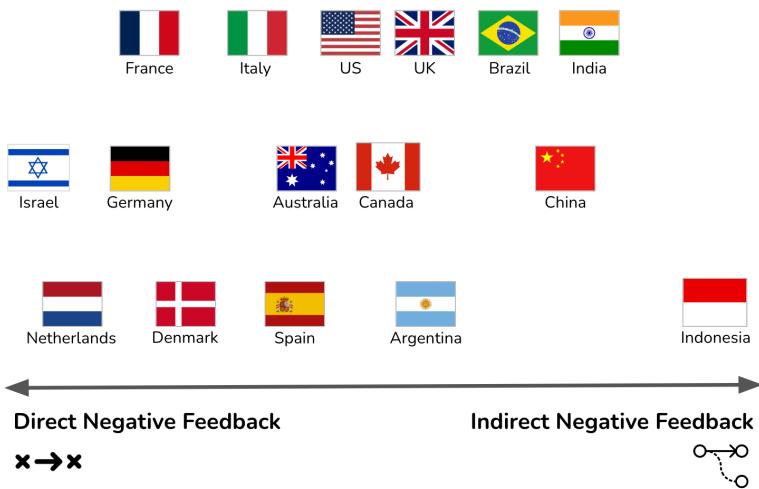
Architects need to **provide constructive criticism** of the plans and ideas of others. All cultures believe that people should give criticism constructively, but the definition of “constructive” varies greatly.



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).

In IT architecture and Software Engineering, providing constructive criticism is essential to ensure that ideas are refined and projects succeed. Here are some examples of issues that relate to architects providing constructive criticism:

- Overly Direct Feedback Leading to Resistance:
  - **Scenario:** An architect from a culture that values directness in feedback reviews a design document created by a team member from a culture that values more diplomatic feedback. The architect bluntly points out flaws in the design, focusing on what is wrong without much positive reinforcement.
  - **Problem:** The team member from the diplomatic culture may perceive the feedback as overly harsh or even disrespectful. Instead of motivating improvement, this direct approach may lead to defensiveness, loss of face, or reluctance to engage in future discussions. Though technically valid, the criticism may not result in the desired improvements due to cultural misalignment.

- Overly Diplomatic Feedback Leading to Misunderstanding:
  - **Scenario:** An architect from a culture that values indirect communication provides feedback on a technical proposal from a team member from a culture that expects more straightforward feedback. The architect couches the criticism positively, saying, “This is a great start, but maybe we could consider some adjustments.”
  - **Problem:** Team members from the direct-feedback culture may not grasp the seriousness of the issues being raised. The subtlety might lead them to underestimate the need for changes, resulting in a final product that falls short of expectations. The feedback is not acted upon as strongly as it should be, leading to potential project setbacks.
- Cultural Differences in Group Settings:
  - **Scenario:** During a design review meeting, an architect from a culture that values group harmony provides criticism very indirectly, using vague terms to avoid singling out any individual. The meeting includes team members from cultures more accustomed to direct feedback in group settings.
  - **Problem:** The more direct cultures may find the feedback unhelpful or confusing, as the indirect criticism doesn’t identify the issues or the individuals responsible for them. This problem could lead to inaction or misunderstandings about what the team must address. On the other hand, if the architect tries to adapt and provides more direct feedback, it might cause discomfort or embarrassment for those not used to such an approach.
- Mixed Reactions in Diverse Teams:
  - **Scenario:** An architect leads a multinational team with members from various cultural backgrounds, including high-context and low-context cultures. During a code review, the architect provides criticism in a balanced way, aiming to be constructive by combining positive feedback with suggestions for improvement.

– **Problem:** The feedback is received differently across the team. Team members from cultures that prefer direct feedback may feel that the criticism is too soft and doesn't address the real issues. In contrast, those from cultures that prefer more diplomatic feedback may feel the criticism is too harsh. This results in confusion and varying degrees of engagement with the feedback, making it difficult for the team to move forward cohesively.

- **Feedback in Written vs. Verbal Form:**

– **Scenario:** An architect from a culture that values written feedback for clarity and record-keeping provides detailed written criticism of a system architecture proposal. The feedback is sent to a team member from a culture that prefers face-to-face communication for sensitive matters.

– **Problem:** The written feedback, although clear and precise, may be perceived as impersonal or even confrontational by the team member, especially if it touches on significant flaws. The lack of a personal touch might lead to misinterpretation of the architect's tone and intent, reducing the effectiveness of the feedback and potentially harming the working relationship.

These examples illustrate how cultural differences in the perception of "constructive" feedback can lead to communication problems in software engineering and IT architecture. Architects must be aware of these differences and adapt their feedback style according to their audience's cultural expectations to ensure that their criticism is both understood and acted upon effectively.

## 19.3: 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.

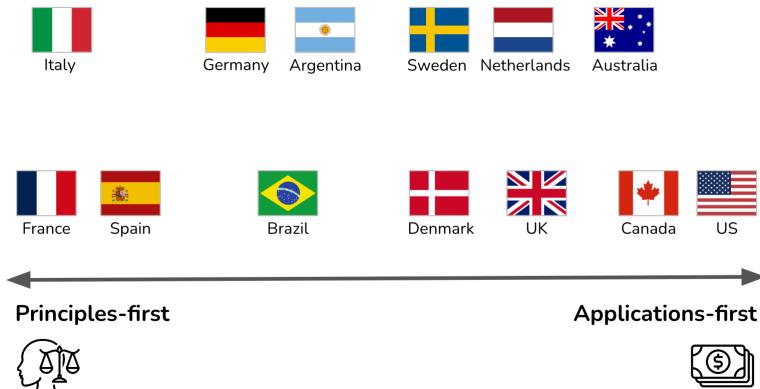


image by fizkes from istock

The Persuading scale assesses how people 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 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 principal 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).*

In IT architecture and Software Engineering, effectively persuading stakeholders is critical for gaining buy-in on decisions and plans. Here are some examples that illustrate communication problems related to the “Persuading” dimension:

- **Mismatch Between Holistic and Specific Thought Patterns:**
  - **Scenario:** An architect from a Western culture presents a detailed, step-by-step plan for a new software architecture to a team that includes members from East Asian cultures. The architect breaks down the argument into distinct components, expecting the team to follow the logic sequentially.
  - **Problem:** The team members from East Asian cultures, who may prefer holistic thinking, might struggle to see how these separate components fit into the broader system. They might find the argument less convincing because it doesn't address the overall harmony and integration of the system from the

outset. As a result, the team meets the architect's proposal with skepticism or requests for a more integrated explanation.

- **Differences in Deductive vs. Inductive Reasoning:**

- **Scenario:** An architect must convince a multinational team to adopt a new technology stack. The architect, coming from a Germanic culture, uses a deductive approach, starting with fundamental principles of software design and building towards the conclusion that the new technology is the best choice.
- **Problem:** Team members from Anglo-Saxon cultures might find this approach tedious and unconvincing because they prefer an inductive approach, where the argument starts with practical examples of successful technology applications. The architect's failure to start with concrete examples may lead to a lack of engagement or difficulty in convincing these stakeholders.

- **C-Level Executives Preferring Applications-First Approach:**

- **Scenario:** The task of an architect is to persuade a group of C-level executives to approve a major overhaul of the company's IT infrastructure. The architect, aware of the executives' preference for brevity, attempts to persuade them using an applications-first approach, highlighting the immediate business benefits and ROI.
- **Problem:** Some executives, particularly those with engineering or technical backgrounds from cultures that value principles-first reasoning, may find this approach insufficient. They may want to see the underlying principles and technical justifications before being convinced, leading to a potential disconnect and hesitation to approve the plan.

- **Persuasion in Multicultural Teams with Varying Expectations:**

- **Scenario:** An architect works with a diverse project team that includes members from holistic and specific-thinking cultures. The architect tries to persuade the team to adopt a new project management tool by explaining its detailed features and benefits in a specific, component-by-component manner.

– **Problem:** The holistic thinkers on the team might find this argument unconvincing because it doesn't address how the tool fits into the broader workflow or organizational goals. On the other hand, the specific thinkers might be satisfied with the details but may miss the overall strategic alignment. The architect's failure to balance both thought patterns results in partial buy-in, with some team members remaining unconvinced.

- **Educational and Philosophical Influences on Persuasion:**

– **Scenario:** An architect trained in a Western analytical tradition tries to persuade a multicultural team about the superiority of microservices architecture by focusing on empirical evidence and logical analysis. This approach is rooted in their educational background, emphasizing critical thinking and empirical validation.

– **Problem:** Team members from cultures with a more conceptual approach to problem-solving might find this evidence-based approach lacking in conceptual depth. They may prefer a discussion that starts with overarching principles or theoretical considerations about system design before delving into the specifics. This mismatch can lead to difficulty in gaining complete consensus.

These examples highlight how differences in cultural backgrounds and reasoning preferences can lead to communication problems when architects attempt to persuade others. To be effective, architects must be adaptable in their persuasion strategies, ensuring that they address both applications-first and principles-first approaches and balance specific and holistic thinking depending on their audience.

## 19.4: Leading

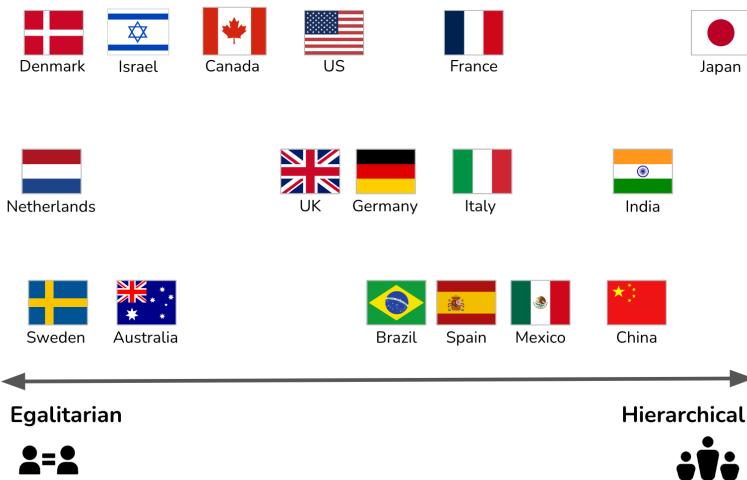
Architects have informal and sometimes formal authority. The leading scale measures the degree of **respect and deference shown to authority figures**.

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 jacoblund from 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).

In IT architecture and Software Engineering, the leadership style of an architect can significantly impact team dynamics and project outcomes. The difference between hierarchical and egalitarian cultures can lead to communication problems and misunderstandings. Here are some examples related to the “Leading” dimension:

- Perception of Weak Leadership in Hierarchical Cultures:
  - **Scenario:** An architect from an egalitarian culture leads a multinational team that includes members from hierarchical cultures. The architect adopts a democratic leadership style, encouraging open discussions and inviting input from all team members before making decisions.
  - **Problem:** Team members from hierarchical cultures may perceive this approach as a lack of decisive leadership. They might expect the architect to make authoritative decisions and provide clear direction. The perceived indecisiveness can lead to frustration, decreased respect for the architect, and inefficiencies, as team members may wait for explicit instructions rather than take the initiative.
- Perception of Dictatorial Leadership in Egalitarian Cultures:

- **Scenario:** An architect from a hierarchical culture leads a project team in an egalitarian environment. The architect takes a top-down approach, making decisions unilaterally and expecting the team to follow instructions without much discussion.
- **Problem:** Team members from the egalitarian culture may see this leadership style as overly authoritarian and stifling. They may feel disempowered and disengaged, leading to reduced creativity and collaboration. The team might perceive the architect as a “dictator,” harming team morale and hindering open communication.

- **Difficulty in Balancing Authority Across Cultures:**

- **Scenario:** An architect manages a global project team with members from hierarchical and egalitarian cultures. The architect balances their leadership approach by being decisive on critical issues while encouraging input and collaboration.
- **Problem:** This mixed approach may lead to confusion within the team. Members from hierarchical cultures still expect more direct leadership and may feel uncertain when decisions are open for discussion. Conversely, members from egalitarian cultures might perceive the decisive moments as undermining their input, leading to dissatisfaction and a sense of inconsistency in leadership. The architect’s attempt to accommodate both styles can result in a perception of inconsistency or lack of clarity in leadership.

- **Challenges in Establishing Authority in Egalitarian Cultures:**

- **Scenario:** An architect from a hierarchical culture joins a team in an egalitarian company and attempts to establish authority by emphasizing their role and making decisions independently.
- **Problem:** In the egalitarian culture, this approach may backfire, as the team expects the architect to lead through consensus and collaboration. The team members may resist or bypass the architect’s decisions, seeking approval or input from peers instead. This problem can lead to tensions and a breakdown in team cohesion, with the architect struggling to assert their authority effectively.

- **Difficulty in Decision-Making Due to Cultural Expectations:**
  - **Scenario:** An architect from an egalitarian culture is leading a design review session with a team of members from both hierarchical and egalitarian cultures. The architect opens the floor for feedback and discussion, intending to decide based on the consensus.
  - **Problem:** Team members from the hierarchical culture might be reluctant to voice their opinions in public, expecting the architect to lead the decision-making. Meanwhile, members from the egalitarian culture might be more vocal, potentially dominating the discussion. This dynamic can lead to an imbalance in contributions, with the final decision not fully reflecting the views of the entire team. The architect might struggle to reconcile these differing expectations, leading to dissatisfaction among some team members.

These examples show how differences in cultural attitudes towards authority and leadership can create communication problems in software engineering and IT architecture. Architects must be aware of these cultural dynamics and engage in open conversations with their teams to set clear expectations about leadership style, ensuring everyone feels respected and their contributions are valued.

## 19.5: Deciding

Architectural work is about [making decisions](#)<sup>3</sup>. The Deciding scale measures the degree to which a culture is **consensus-minded**.



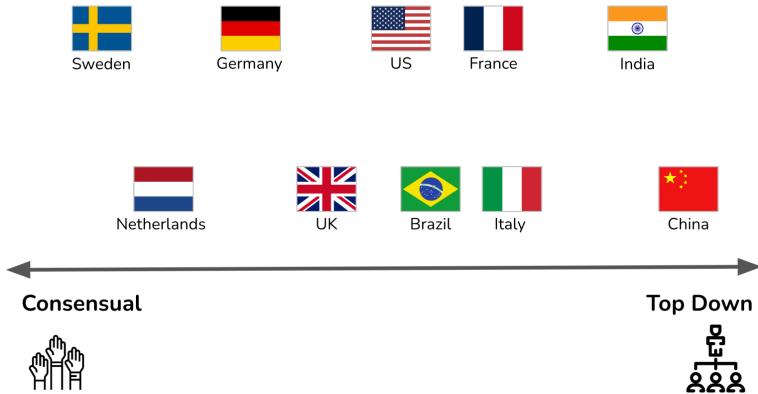
image by fizkes from istock

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 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.

---

<sup>3</sup><https://architectelevator.com/gregors-law/>



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

Decision-making is a critical part of the job in IT architecture and Software Engineering. Here are some examples that illustrate the challenges associated with the “Deciding” dimension:

- Conflict Between Consensus and Hierarchical Decision-Making:
  - **Scenario:** An architect from a hierarchical culture is leading a project team with members from both hierarchical and consensus-oriented cultures, including team members from different parts of Europe and the United States. The architect is used to making decisions after consulting with a few key stakeholders but expects to have the final say.
  - **Problem:** The European team members expect a more consensus-driven approach, where decisions are made only after thorough discussion and agreement from the entire group. Meanwhile, the American team members might be more comfortable with the architect making a quick, top-down decision after considering input. The differing expectations lead to frustration: some members feel excluded when decisions are made too quickly, and others may see the process as unnecessarily slow and cumbersome. This results in delays and dissatisfaction within the team.

- **Delayed Decision-Making Due to Consensus Expectations:**

- **Scenario:** An architect is working with a team, where decisions are typically made through a slow, consensus-driven process that involves input from all levels of the hierarchy. The architect, coming from a culture that values quicker decision-making, becomes frustrated with the time it takes to reach an agreement.
  - **Problem:** The architect's attempts to expedite the decision-making process are met with resistance, as the Japanese team members are uncomfortable deciding without complete consensus. This problem leads to delays in the project as the architect struggles to reconcile the need for timely decisions with the team's cultural expectation for consensus. The architect may inadvertently cause tension by pushing for a decision before the team is ready.

- **Decision-Making Deadlock in a Culturally Diverse Team:**

- **Scenario:** An architect leads a multicultural team tasked with selecting a new technology stack. The team includes members from different parts of Europe (consensus-oriented, egalitarian), Asia (hierarchical, consensus-oriented), and the United States (individualistic, often leader-driven). The architect attempts to reach a consensus but finds discussions drag on without a clear decision.
  - **Problem:** Some team members are reluctant to finalize a decision without complete group agreement, while others expect the architect to step in and make a decisive call when discussions stall. The architect, unsure how to proceed, may either push through a decision, alienating some team members, or allow the discussions to continue indefinitely, frustrating others who expect a quicker resolution. This problem leads to a deadlock, with the project stalling due to the inability to make timely decisions.

- **Perception of Indecisiveness Due to Consensus-Building:**

- **Scenario:** An architect from a consensus-driven culture is working with a team from a culture that values quick, top-down decision-making. The architect spends considerable

time gathering input and seeking agreement from all stakeholders before deciding on a critical architectural change.

- **Problem:** The team members perceive the architect's careful consensus-building approach as indecisiveness or lacking leadership. They may become frustrated with the slow pace and lose confidence in the architect's ability to lead the project effectively. This problem can result in a lack of cohesion, with team members potentially bypassing the architect to push for quicker decisions through other channels.

- **Frustration Over Unilateral Decision-Making:**

- **Scenario:** An architect from a culture where leaders are expected to make unilateral decisions is working with a team that prefers consensus-driven decision-making. The architect, believing it's their responsibility to make the final call, decides on a key architecture component without extensive team consultation.
- **Problem:** The team members feel alienated and disrespected because they expect to be involved in decision-making. They may view the architect as autocratic and resist or even sabotage the decision by not fully supporting its implementation. This problem creates friction and reduces team collaboration, ultimately harming the project's success.

These examples show how differing cultural expectations around decision-making can lead to significant communication problems in software engineering and IT architecture. Architects must recognize these cultural differences and actively manage decision-making processes to ensure that all team members feel involved and respected while keeping the project on track. Open conversations about decision-making expectations are crucial to navigating these challenges effectively.

## 19.6: 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).

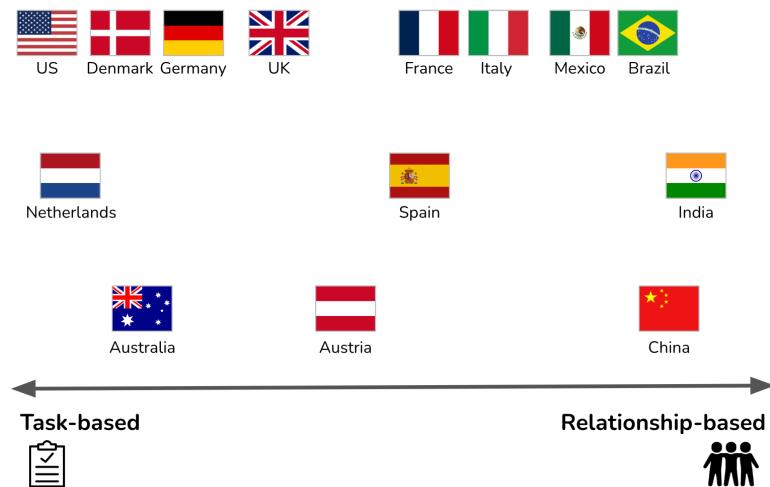


image by scyther5 from istock

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).

In IT architecture and Software Engineering, building trust with stakeholders is essential for the success of projects. Here are some examples that illustrate challenges related to the “Trusting” dimension:

- **Task-Based vs. Relationship-Based Trust Building:**
  - **Scenario:** An architect from a task-based culture is assigned to lead a multinational team that includes members from a relationship-based culture. The architect focuses on delivering high-quality work and proving reliability through meeting deadlines and technical competence, assuming this will build trust with the team.
  - **Problem:** Team members from the relationship-based culture might find the architect distant or impersonal. They may expect more personal interactions, such as casual conversations, shared meals, or social gatherings, to establish trust. The lack of effort in building a personal connection may lead to mistrust or disengagement from the team, who might not fully support the architect's decisions or collaborate effectively.
- **Misalignment in Trust-Building Expectations with Clients:**

- **Scenario:** An architect from a relationship-based culture works with a client from a task-based culture. The architect invests time in getting to know the client personally, sharing meals, and engaging in small talk, believing this will build a strong foundation of trust.
  - **Problem:** The client, who values task-based trust, may perceive these efforts as unnecessary or a waste of time. They might feel that the architect is not focused enough on delivering results and might become frustrated with what they see as a lack of professionalism. This problem can lead to a strained relationship, with the client doubting the architect's ability to deliver on technical promises.
- **Difficulty in Integrating into a Relationship-Based Team:**
    - **Scenario:** An architect from a task-based culture joins a team in a company that operates within a relationship-based culture. The architect is eager to start working on the project and skips social events, such as team lunches or informal gatherings, to focus on technical tasks.
    - **Problem:** The team may view the architect as aloof or uninterested in building a personal connection, which is crucial in their culture for establishing trust. As a result, the team might be less willing to collaborate openly, share information, or support the architect's initiatives. This problem leads to a lack of cohesion and potentially undermines the project's success.
  - **Challenges in Gaining Stakeholder Trust Across Cultures:**
    - **Scenario:** An architect works with stakeholders from task- and relationship-based cultures. The architect prioritizes building trust with the task-based stakeholders by consistently delivering on project milestones while attempting to build trust with relationship-based stakeholders through regular social interactions and personal engagement.
    - **Problem:** The architect struggles to balance these approaches, potentially leading to dissatisfaction on both sides. Task-based stakeholders may feel that the architect is spending too much time on "soft" activities, while relationship-based

stakeholders might feel neglected if the architect focuses too heavily on task delivery. If not well-managed, this dual approach can lead to confusion and weakened trust between both groups.

- **Erosion of Trust Due to Cultural Misunderstandings:**

- **Scenario:** An architect from a task-based culture is working with a team from a relationship-based culture. During a critical project phase, the architect skips a planned social event due to workload, believing that delivering the project on time is the most crucial way to build trust.
- **Problem:** The team from the relationship-based culture may perceive the architect's absence as a lack of respect or commitment to the team's relationship, leading to disappointment and erosion of trust. They might start questioning the architect's intentions or become less cooperative, which could affect team morale and the project's overall progress.

These examples highlight how differences in trust-building approaches can lead to communication problems in software engineering and IT architecture. Architects need to be mindful of cultural differences and adapt their methods to build trust effectively with stakeholders from various backgrounds. This might involve a combination of delivering reliable work and investing time in personal relationships, depending on the cultural context of the team or stakeholders involved.

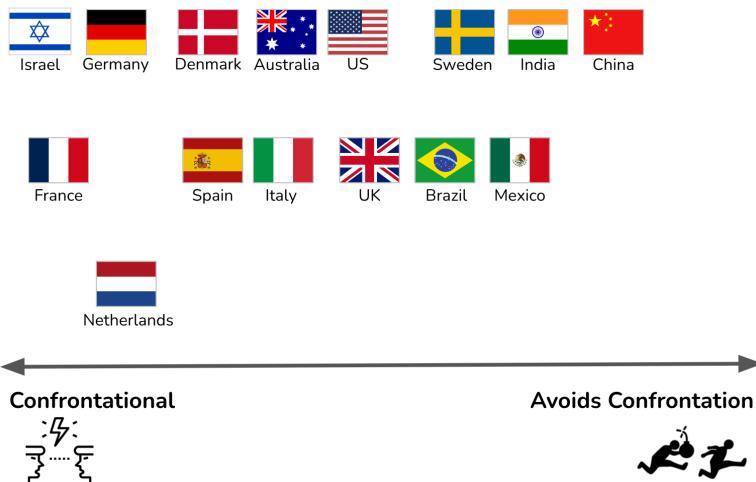
## 19.7: Disagreeing

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. The Disagreeing scale measures **tolerance for open debate** and inclination to see it as helpful or harmful to collegial relationships (Figure 7).



image by fizkes from 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).*

In Software Engineering and IT Architecture, disagreements are inevitable, especially when dealing with complex projects and diverse teams. Here are some examples related to the “Disagreeing” dimension:

- Perceived Aggressiveness in Direct Disagreement:
  - **Scenario:** An architect from a culture that values direct communication and open debate is leading a discussion on a new system design with a team that includes members from cultures that avoid confrontation. The architect openly challenges ideas and expects others to do the same, believing this challenge will lead to the best outcome.
  - **Problem:** Team members from cultures that view open disagreement as disruptive may find the architect’s direct approach aggressive or rude. They might withdraw from the discussion, avoid sharing their opinions, or feel uncomfortable contributing, which can stifle creativity and collaboration. This problem could lead to a lack of diverse input in the design process and potentially poorer decisions.
- Avoidance of Critical Feedback:

- **Scenario:** An architect from a culture that prefers to avoid open conflict works with a team that sees disagreement as a healthy part of decision-making. When presented with a flawed design, the architect hesitates, openly criticizing the work and offering indirect feedback that the design “might need some improvement.”
- **Problem:** The German team members might interpret the architect’s indirect feedback as approval or as a sign that the issues are minor. As a result, they may proceed with the flawed design, unaware of the architect’s true concerns. This can lead to significant problems later in the project when the issues become more apparent, and more costly to fix. The architect’s reluctance to engage in open disagreement may lead to misunderstandings and suboptimal outcomes.

- **Misinterpretation of Passionate Debate:**

- **Scenario:** An architect from a culture that embraces passionate debate engages in a heated discussion with colleagues who value harmony and consensus. The architect raises their voice and uses strong language to emphasize their points, a sign of commitment and engagement.
- **Problem:** The Malaysian team members may interpret the passionate debate as anger or personal conflict, which they might find unsettling. They may avoid further confrontation or seek to smooth over the disagreement without fully addressing the issues, leading to unresolved conflicts. The architect’s approach, while intended to spark productive debate, might instead lead to discomfort and avoidance of future discussions.

- **Challenges in Reaching Consensus:**

- **Scenario:** An architect from a culture that values consensus and group harmony is working with a team from a culture that is comfortable with open disagreement and individual opinions. During a meeting, the architect avoids voicing disagreement with a proposed solution, preferring to seek a behind-the-scenes resolution.

- **Problem:** The American team members might perceive the lack of open disagreement as agreement or indifference, leading them to push forward with the solution without further discussion. Later, when issues arise, the architect's unvoiced concerns might surface, causing frustration among the team members who thought the decision had been fully supported. The differing approaches to disagreement can result in misalignment and project delays.
- **Conflict Avoidance Leading to Poor Decision-Making:**
  - **Scenario:** An architect is leading a multicultural team where some members are from cultures that highly value avoiding conflict and others from cultures that see conflict as a natural part of decision-making. The architect, aware of the cultural differences, tries to avoid open disagreements to keep the team harmonious.
  - **Problem:** Critical issues and differing opinions may not be fully explored by avoiding confrontation, leading to decisions that do not consider all perspectives. Some team members might feel that important debates are being stifled, leading to dissatisfaction and disengagement. Meanwhile, the team members who prefer to avoid conflict may feel uncomfortable if disagreements are forced. This problem can result in decisions that are not fully vetted, potentially leading to issues later in the project.

These examples illustrate how different cultural attitudes towards disagreement can lead to communication challenges in software engineering and IT architecture. Architects must navigate these differences carefully, finding ways to encourage constructive debate without alienating team members or stifling essential discussions. Open conversations about handling disagreements help align expectations and improve team dynamics.

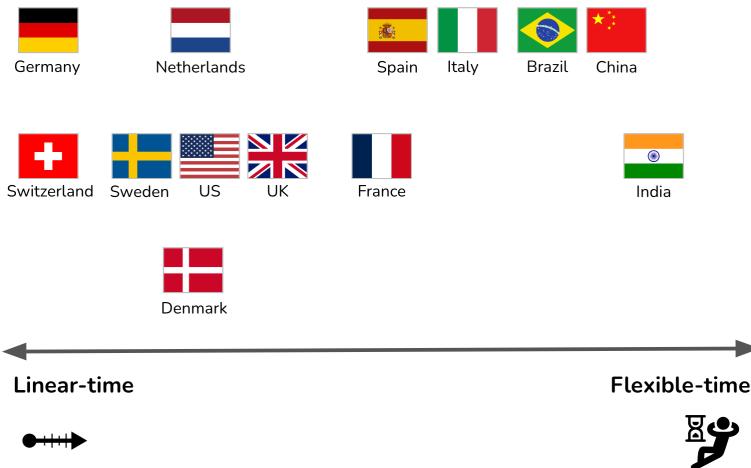
## 19.8: Scheduling

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 bobex\_73 from 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).

In IT architecture and Software Engineering, scheduling and time management are critical, but different cultural attitudes toward time can lead to communication problems. Here are some examples related to the “Scheduling” dimension:

- **Tension Due to Strict Adherence to Schedules:**
  - **Scenario:** An architect from a monochronic culture, where punctuality and strict adherence to schedules is highly valued, is leading a project with a team that includes members from a polychronic culture, where schedules are more flexible and interruptions are common.
  - **Problem:** The architect becomes frustrated when team members frequently arrive late to meetings or discussions veer off the planned agenda. The polychronic team members perceive the architect's insistence on sticking to the schedule as inflexible or overly rigid. This tension can create an uncomfortable working environment, with the architect feeling disrespected and the team feeling pressured and misunderstood.
- **Project Delays Due to Flexible Scheduling Attitudes:**

- **Scenario:** An architect from a polychronic culture, where multitasking and adjusting plans based on the situation are everyday, is managing a project for a client from a monochronic culture. The architect is comfortable changing the project timeline as new tasks emerge or priorities shift.
- **Problem:** The client, who expects a linear and predictable schedule, becomes concerned when deadlines are missed or the project timeline seems fluid. They perceive the architect as disorganized or unprofessional, leading to a loss of trust. The architect's flexible approach to scheduling conflicts with the client's expectations, resulting in dissatisfaction and potential conflicts over project management practices.

- **Misalignment in Meeting Expectations:**

- **Scenario:** An architect in a multicultural team sets up regular meetings to review project progress. The team includes members from both monochronic cultures and polychronic cultures. The architect expects everyone to arrive on time, follow the agenda closely, and finish the meeting within the scheduled time.
- **Problem:** Team members from the polychronic culture might arrive late, extend discussions beyond the agenda, or prioritize ongoing tasks over the meeting. This problem frustrates the monochronic team members who value punctuality and structure. The meeting becomes less productive, with some members feeling their time is not respected, while others feel rushed or constrained by the rigid schedule.

- **Challenges in Coordinating Cross-Cultural Project Teams:**

- **Scenario:** An architect is coordinating a project involving teams from both monochronic and polychronic cultures. The project requires collaboration across multiple time zones, with some teams expecting strict adherence to the project timeline and others seeing the schedule as more of a guideline.
- **Problem:** The architect struggles to keep the project on track because the teams have different expectations about deadlines and meeting times. The monochronic teams may become frustrated when deliverables are delayed due to the more flexible approach of the polychronic teams, leading to friction and

blame-shifting. The project risks falling behind schedule as the architect tries to balance these differing cultural attitudes toward time management.

- **Perception of Disrespect Due to Scheduling Flexibility:**

- **Scenario:** An architect from a monochronic culture is working with a vendor from a polychronic culture to implement a new software system. The architect sends detailed schedules and expects the vendor to follow them closely. However, the vendor often changes meeting times or shifts deadlines based on other priorities.
- **Problem:** The architect perceives the vendor's behavior as disrespectful and unprofessional, leading to frustration and a strained relationship. On the other hand, the vendor sees flexibility as a regular part of managing multiple clients and projects simultaneously. The difference in scheduling expectations leads to misunderstandings and decreased collaboration effectiveness.

These examples highlight the need for open communication about scheduling preferences in software engineering and IT architecture. Architects can foster a more harmonious and productive working environment by recognizing and respecting differing cultural attitudes towards scheduling and finding ways to bridge the gap, such as setting clear expectations and being flexible where possible. This emphasis on open communication encourages us to share our scheduling preferences and understand those of our colleagues, creating a supportive and collaborative work environment.

## 19.9: 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. They 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.

## 19.10: 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?*

# 20: The Human Side of Decision-Making



image by metamorworks from 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.

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. 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/>

## 20.1: Understanding Human Biases and Limitations

Fundamental biases influencing decision-making include outcome, hindsight, and confirmation biases.

### 20.1.1: 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 decided in the first place. Results matter, but if you trip over an unlucky result and think you picked the wrong option, you're learning to fail with flair. This faulty thinking could make you make even wackier choices in the future.



image by mesh

cube from 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. 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.

Suppose we only focus on the outcome, neglecting the rich backstory and the information available at the time of the decision. In that case, 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**

- Due to time constraints, a software team decided to release a critical update without performing adequate regression testing. The update has been released, and fortunately, no major issues have been 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 has gained popularity and has not encountered 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: 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 critical quality control step. Future projects might suffer from hidden bugs and poor code quality, which could have been caught during code reviews.

#### Example 5: 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 essential for teams to critically evaluate their methodologies and ensure that success is attributed to sound practices rather than favorable outcomes alone.

### 20.1.2: 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.



image by designer491 from istock

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 **write 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.

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: 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 tech-

nology 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, the perceived advantages of the stack, and the extensive procurement process.

#### **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: 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 that other pressing issues and constraints influenced the original decision-making process.

#### **Example 4: 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 claimed that the integration issues were obvious and should have been anticipated. They overlooked that the integration appeared straightforward at the time of decision-making, and the issues were not foreseeable with the information available.

#### **Example 5: Feature Failure**

- A new feature is released but fails to gain user adoption, which 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 6: 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 7: 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.

By acknowledging the role of hindsight bias, teams can foster a more realistic and fair evaluation of past projects and decisions.

### **20.1.3: 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.



image by designer491 from istock

Awareness of this sneaky bias is essential because your brain loves playing 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 be as helpful 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.

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.

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: 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 supporting the decision while disregarding case studies or reports highlighting challenges and failures associated with the new technology. This can lead to underestimating the risks and difficulties of the adoption process.

#### **Example 3: 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 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 strongly believes 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: 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 7: 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 likely to elicit positive responses, thus reinforcing their belief that the feature is well-received.

To mitigate the impact of confirmation bias, teams should actively seek out and consider disconfirming evidence, adopt a critical thinking approach, and encourage diverse perspectives. By being aware of this bias, teams can make more balanced and informed decisions, ultimately leading to better software outcomes.

### 20.1.4: Other Human Limitations Influencing Decision-Making

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 aaronamat from 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!

## 20.2: Understanding Power and Limitations of Human Intuition in Decision-Making

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>. But our unawareness of such knowledge does not mean we cannot use it.

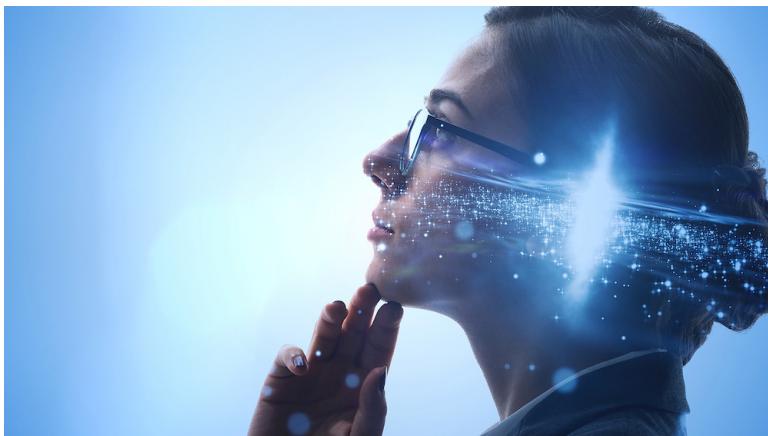


image by metamorworks from istock

One of the main advantages of intuition in decision-making 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.

Intuition is especially appropriate under certain conditions:

- **Time Pressure:** When time is limited and a detailed analysis isn't feasible, intuition can guide you when otherwise you would be stuck.

---

<sup>4</sup><https://www.amazon.com/Software-Creativity-2-0-Robert-Glass/dp/0977213315>

- **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 when more effort is warranted, including those with ample time, high importance, lack of expertise, and the possibility to use a structured decision-making process.

Intuition in IT can be a **powerful tool** when 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.

### 20.2.1: 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 apparent 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:** Based on this intuition, the engineer quickly narrows down the potential causes, focusing on recent changes in the configuration files.
3. **Quick Resolution:** This intuition-driven approach reveals a misconfiguration in 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.

### **20.2.2: 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 they have used.

#### **Bad Use of Intuition**

1. **Ignoring Data:** The team lead dismisses team members' concerns and data 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 can lead to suboptimal decisions when used without adequate consideration of data and other perspectives.

Balancing intuition with data-driven decision-making and collaborative input often leads to the best outcomes in software engineering.

## 20.3: Understanding Human Indecisiveness

We frequently fall into the indecisiveness trap. Why? Well, people can be indecisive for all sorts of amusing reasons.

### 20.3.1: Delaying Decisions

Many 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 without ice cream. Delaying, postponing, or deprioritizing the decision-making process is an implicit choice.



image by dragon claws from istock

Dodging decisions often lead to significant consequences, even if they aren't immediately obvious. Here are a few examples:

#### Microservices vs. Monolithic Architecture

- **Scenario:** A team is debating whether to move to a microservices architecture or continue with their existing monolithic system.

Rather than making a decision, they keep delaying it, hoping that some perfect clarity will emerge.

- **Implicit Decision:** The team effectively “decides” to stay with their monolithic system by not choosing. Over time, this makes it harder to scale, introduces more technical debt, and slows deployment and innovation. They’ve made a choice—whether they realize it or not.

## Code Refactoring vs. Feature Development

- **Scenario:** Engineers know that a particular part of the codebase needs refactoring, but with each sprint, the refactoring task gets postponed in favor of new features.
- **Implicit Decision:** By deprioritizing refactoring, the team implicitly decides to live with a growing pile of technical debt, leading to more bugs, slower performance, and increasingly difficult maintenance.

## Cloud Migration

- **Scenario:** A company wants to move its infrastructure to the cloud but keeps postponing the decision, citing the need for further research or budgetary constraints.
- **Implicit Decision:** By delaying the migration, they implicitly decide to continue using outdated on-premise systems, which may be less efficient and more expensive in the long run. They also miss out on the agility and scalability benefits of cloud infrastructure.

## Testing Strategy

- **Scenario:** A team debates about the scope of automated vs. manual testing. However, because they can’t decide, they continue relying on an inconsistent mix of both without a comprehensive strategy.
- **Implicit Decision:** The lack of a solid testing strategy leads to reduced code quality, longer release cycles, and more defects slipping into production. The choice to not formalize testing becomes a costly implicit decision.

## Single Sign-On (SSO) Integration

- **Scenario:** A company debates whether to implement SSO for its internal systems to streamline user authentication. However, security and engineering teams defer the decision to focus on more “urgent” projects.
- **Implicit Decision:** By not implementing SSO, the company has multiple login systems, causing frustration for users and increased security risks due to inconsistent authentication methods.

In all of these cases, failing to make a decision is itself a decision—and often a costly one. The key takeaway here is that avoiding decisions in technical scenarios can lead to significant consequences, even if they aren’t immediately obvious.

### 20.3.2: Overwhelmed by Numerous Decisions

Then there’s the classic “too many choices” dilemma. When faced with an array 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 simultaneously. Suppose we spend too much mental energy deciding what color to paint the break room. In that case, we might leave no brainpower for the big decisions, like how to keep the servers from catching fire.



image by cyano66 from istock

The “too many choices” dilemma is a common problem where teams or decision-makers become overwhelmed by many low-priority decisions, often detracting from critical, high-impact choices. Here are a few examples:

### Library and Dependency Updates

- **Scenario:** Developers ask the IT architect to review and approve every minor version upgrade for third-party libraries, even those that involve minimal risk or functionality changes.
- **Problem:** By focusing the architect’s attention on these low-impact updates, critical decisions (like system scalability, security measures, or architectural design) may get delayed. It’s like asking the architect to approve every paint stroke while the house’s foundation needs reinforcement.

### Code Formatting Standards

- **Scenario:** The engineering team spends excessive time debating the best code formatting or style guidelines, like whether to use 2 or 4 spaces for indentation.
- **Problem:** While consistency in code formatting is important, over-allocating time to decisions like these diverts attention from more significant decisions, such as improving the application’s architecture or fixing major bugs. It’s like worrying about the snacks in the break room while production servers are down.

### Platform or Tool Selection

- **Scenario:** The team chooses between different code editor plugins, build tools and minor productivity tools. Each option is reviewed in detail, with countless meetings to compare feature lists.
- **Problem:** Overanalyzing small tooling decisions causes unnecessary delays and burns mental energy that is better spent on high-impact decisions, such as selecting a cloud provider or deciding the proper microservices orchestration approach.

### Server Naming Conventions

- **Scenario:** The IT department holds extensive discussions on naming the new servers or virtual machines in the cloud environment—should they use city names, numbers, or specific animal species?
- **Problem:** This seemingly minor debate consumes valuable time and energy, while more critical infrastructure decisions, like disaster recovery planning or server load balancing, get delayed or ignored.

### Approving Minor Configuration Changes

- **Scenario:** IT architects are asked to approve every minor configuration tweak, such as adjusting the timeout for a service or tweaking memory limits for non-critical applications.
- **Problem:** The architects drown in trivial decisions, leaving them little bandwidth to focus on higher-priority issues, like designing robust, scalable system architectures or improving application security.

In these examples, the team's limited cognitive capacity is consumed by trivial choices, leaving insufficient mental energy for critical decisions that genuinely impact system performance, security, or scalability. This issue highlights the importance of delegating or automating low-priority decisions and reserving strategic brainpower for what truly matters.

### 20.3.3: Emotions and Grief

Indecisiveness also loves to rear its head when emotions are involved, especially when all options are 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.



image by violetastoimenova from istock

Teams often face situations where every option feels undesirable, and indecisiveness can creep in due to emotional frustration or the hope for a perfect solution. Here are some examples:

### Legacy System Migration

- **Scenario:** A company runs critical applications on a legacy system that is slow, costly to maintain, and increasingly prone to failure. The options are to either:
  1. Perform a risky, costly, and time-consuming migration to a modern system.
  2. Continue investing in patching the legacy system, with the risk of a major outage.
- **Emotional Factor:** No one wants to be responsible for the potential failures of a complex migration, but continuing with the legacy system feels like kicking the can down the road.
- **Least Bad Option:** After evaluating the long-term costs and risks, the team decides to proceed with the migration, accepting the immediate pain of potential downtime and cost over the unsustainable alternative of maintaining the outdated system. It's not a great choice, but it's the least bad one, so they bite the bullet.

## Technical Debt Management

- **Scenario:** A project has accumulated significant technical debt. The options are:
  1. Allocate several sprints to pay the debt, delaying new feature development.
  2. Continue adding features on top of a shaky codebase, which will inevitably slow development and increase the risk of defects.
- **Emotional Factor:** Teams often feel frustrated by technical debt, wishing it would somehow resolve itself. However, neither option is appealing—both involve trade-offs between immediate feature delivery and long-term stability.
- **Least Bad Option:** Recognizing that continued development on a brittle foundation will lead to far greater problems, the team accepts the frustration of delaying features and tackles the technical debt, understanding it's the least bad way to ensure future scalability and maintainability.

## Vendor Lock-In

- **Scenario:** A company is heavily invested in a particular cloud provider but realizes the cost is much higher than anticipated. The choices are:
  1. Stay with the current provider and absorb the higher costs.
  2. Undertake a complex and costly migration to a new provider, risking downtime and re-engineering efforts.
- **Emotional Factor:** Switching vendors feels overwhelming and risky, while staying locked into an expensive provider feels frustrating. Hoping for a “magic” solution (e.g., the provider reducing costs) isn’t realistic.
- **Least Bad Option:** After weighing the costs and risks, the company decides to stay with the current vendor, accepting the cost over the uncertainty of migration. It’s not ideal, but it’s the least bad choice given the alternatives.

## Monolithic vs. Microservices Split

- **Scenario:** A monolithic application has grown too large, and the team knows it needs to break it into microservices. The options are:
  1. Undertake a complex, high-risk rewrite of the monolithic application into microservices.
  2. Continue maintaining the monolith, accepting slower release cycles and scaling limitations.
- **Emotional Factor:** Rewriting a system always feels daunting, and no one wants to own the risk of disrupting the system during the transition. However, maintaining the monolith comes with its own set of frustrations, especially as it grows.
- **Least Bad Option:** Recognizing that continuing with the monolith will lead to stagnation, the team opts for a phased approach to break the system into microservices. It's a difficult and risky path, but the alternative of doing nothing is worse.

## Security Fixes vs. Feature Development

- **Scenario:** A critical vulnerability is discovered in an application, but fixing it will require a full sprint of engineering time, delaying a high-priority feature release.
  1. Delay the feature and fix the vulnerability immediately, which will frustrate stakeholders and users awaiting the feature.
  2. Continue with the feature release and address the security issue later, increasing the risk of a breach.
- **Emotional Factor:** The team is pressured by stakeholders eager for new features, but the security risk is causing anxiety. Neither option is appealing, as both involve security and feature delivery trade-offs.
- **Least Bad Option:** The team addresses the security issue first, accepting the frustration of delaying the feature because the consequences of a breach are far worse. It's not a great situation but the least bad decision.

In each case, the choices are less than ideal, and emotions like frustration, anxiety, or fear of failure can cloud judgment. However, teams can move forward with clarity and purpose by acknowledging that no perfect option exists and focusing on the least detrimental path.

## 20.4: Understanding Group Dynamics in Decision-Making

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 prostock\_studio from 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 an individual's extreme tendencies and compensate for human limitations like fatigue.

### 20.4.1: Characteristics of Group-Decision Making

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. 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. Awareness 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.

#### 20.4.2: Examples

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:

#### **20.4.2.1: 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.

#### **20.4.2.2: 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 implementing measures.

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.

#### **20.4.2.3: 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.

#### **20.4.2.4: Example 4: Conflict Resolution in Architecture Decisions**

**Scenario:** The development team is divided over whether to build a new application using a microservices or 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 thoroughly discussing and considering 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. Combining the strengths of different viewpoints 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 choices more effectively, leading to better outcomes and stronger team cohesion.

## 20.5: 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?
- Can you identify any habits or emotional factors contributing to your indecisiveness? What strategies can you employ to overcome these challenges?
- 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?

## 21: Effortless Architecture



image by chinnapong from istock

**IN THIS SECTION, YOU WILL:** Get a summary of lessons learned from Greg McKeown's book *Effortless* on how to functionally structure your work to make the essential activities the easiest ones to achieve.

**KEY POINTS:**

- Greg McKeown's "Effortless: Make It Easier to Do What Matters Most" advocates for a paradigm shift from hard work to smart, effective work by simplifying tasks and processes.
- Key principles include prioritizing important tasks, leveraging automation, and embracing a mindset that values ease and enjoyment in work.
- Greg McKeown's book offers invaluable insights that are particularly relevant for IT architects and software engineers. McKeown's emphasis on simplifying tasks and processes is crucial in the tech industry, where complexity often dominates.

Greg McKeown's "**Effortless: Make It Easier to Do What Matters Most**"<sup>1</sup> advocates for a paradigm shift from hard work to innovative, effective work by simplifying self-created complicated tasks and processes. The book emphasizes achieving goals with minimal strain by fostering an effortless state characterized by clarity and focus. Fundamental principles include prioritizing essential tasks, leveraging automation, and embracing a mindset that values ease and enjoyment in work. McKeown provides practical strategies for reducing unnecessary effort, enhancing productivity, and maintaining sustainable high performance, ultimately enabling individuals to achieve better results with less effort and stress.

I see the Effortless books as a perfect complement to Fred Brooks' essay "**No Silver Bullet**".<sup>2</sup> Fred Brooks posits that no single technological breakthrough will dramatically improve software development productivity because of the inherent, **essential complexity** of the tasks involved. In contrast, Greg McKeown emphasizes the importance of reducing **accidental complexity**—those unnecessary complications we can eliminate to streamline processes and simplify tasks we complicate ourselves. While Brooks highlights the unavoidable challenges intrinsic to software development, McKeown offers a crucial reminder that streamlining and optimizing workflows can significantly reduce extraneous difficulties,

---

<sup>1</sup><https://gregmckeown.com/books/effortless/>

<sup>2</sup>[https://en.wikipedia.org/wiki/No\\_Silver\\_Bullet](https://en.wikipedia.org/wiki/No_Silver_Bullet)

thus enhancing overall efficiency and effectiveness. Or, as McKeown put it "*life doesn't have to be as hard and complicated as we make it.*"

## 21.1: IT Doesn't Have To Be As Hard and Complicated As We Make It

One of the main tasks of architects is to remind everyone that our technical designs, products, and organizational structures don't have to be as complex and complicated as we make them. A fantastic example of this comes from Pragmatic Dave Thomas (I heard this anecdote on the [SE-Radio podcast with Neal Ford<sup>3</sup>](#), ~32 minutes in). A company had problems with its mail post not getting to the proper departments. It wanted a complex optical-character recognition (OCR) system for routing mail posts. However, Dave Thomas suggested using simple and cheap colored envelopes instead. The company did not need to invest millions in building a complex software system with machine-learning capabilities; it solved the problem in a few weeks and saved a lot of money.

Similarly, Greg McKeown's book summarizes many well-known practices into a pragmatic framework with a mindset of effortlessness. As such, I have found that it offers a fresh look at the daily practice of IT architects and software engineers:

- **Simplifying tasks** and processes is crucial in the tech industry, where complexity often dominates. Effortless principles align closely with critical system design and coding practices, such as modular design, clean code, and lean architecture. IT professionals can significantly enhance efficiency by breaking complex systems into manageable modules, writing maintainable code, and focusing on essential features without over-engineering.
- **Prioritization**, another core aspect of McKeown's book, is vital in the fast-paced IT industry. Effectively prioritizing tasks can dramatically impact project outcomes, helping professionals focus on what truly matters. This prioritization leads to more effective project management, resource optimization, and strategic planning, aligning development efforts with business goals.
- **Efficiency and productivity** are also central themes in "Effortless." For software engineers, this translates to practices like automated testing and deployment through CI/CD pipelines, optimization

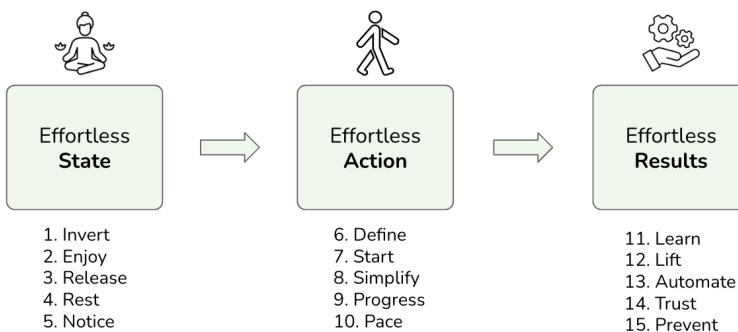
---

<sup>3</sup><https://se-radio.net/2017/04/se-radio-episode-287-success-skills-for-architects-with-neil-ford/>

techniques to enhance code performance, and using development tools that streamline workflows.

- McKeown's advocacy for shifting from hard work to smart work is particularly pertinent in the tech world. This mindset shift promotes **continuous learning**, a healthy **work-life balance**, and **resilience strategies** to handle challenges positively.
- **Collaboration and communication**, highlighted in "Effortless," are essential for IT professionals. They enhance collaboration between development, operations, and business teams and ensure stakeholder engagement, leading to more aligned and informed teams.
- In high-pressure environments like the tech industry, **managing stress** is crucial. McKeown's strategies for reducing unnecessary effort and anxiety can help IT professionals focus on high-value tasks, improve mental health, and boost team morale, creating a more enjoyable work experience.

In the following sections, I will review critical advice from McKeown's work, grouped into Effortless State, Effortless Action, and Effortless Results (Figure 1).



**Figure 1:** The McKeown's Effortless framework: Effortless State, Effortless Action, and Effortless Results.

## 21.2: Effortless State

Many have encountered the Effortless State, a **peak experience** when their physical, emotional, and mental well-being align perfectly. You feel physically rested, emotionally unburdened, and mentally energized in this state. You become entirely aware, alert, present, attentive, and focused on what's important. This state allows you to **concentrate on what matters** more quickly and efficiently (Figure 2).

When you achieve the Effortless State, it's a sign that **your brain is operating at its full capacity**. In this optimal condition, tasks that usually feel difficult become significantly easier. You can navigate challenges with a sense of flow and clarity, making decisions and performing actions with a heightened **sense of purpose and precision**. This state enhances your productivity and opens up new avenues for personal growth and development.

However, reaching and maintaining this state can be impeded by **mental clutter**. Clutter can take many forms, including outdated assumptions, negative emotions, and toxic thought patterns. These mental obstacles drain your cognitive resources and make everything feel more complicated than it should be. By clearing this clutter and fostering a supportive mental environment, you can unlock your brain's full potential and access the Effortless State consistently.



**Figure 2:** Effortless State is a part of the McKeown's Effortless framework (State, Action, Results).

### 21.2.1: 1. INVERT: What If This Could Be Easy?

Feeling overwhelmed is often not due to a situation's inherent complexity but because we are **overcomplicating it in our minds**. Instead of asking, “*Why is this so hard?*” McKeown proposes to invert the question by asking, “*What if this could be easy?*” Challenge the assumption that the “right” way is necessarily harder. When faced with overwhelming tasks, ask yourself, “*How am I making this harder than it needs to be?*”

By asking, “*What if this could be easy?*” you can reset your thinking. Or, as Kent Beck famously stated, for each desired change, **make the change easy, then make the easy change**.

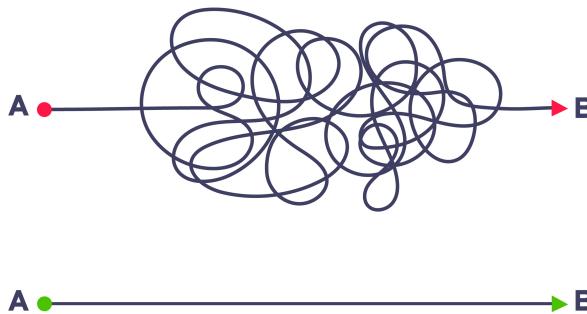


image by mironov konstantin from istock

IT and software engineering have integrated the principles of effortless inversion, transforming many complex tasks into efficient processes. But we must always remind ourselves not to overthink our work.

Here are a few examples that IT architects can use as an inspiration to help their organizations make things easier:

- **Simplifying Code Maintenance With Clean Design:** Rushing to build new features by creating shortcuts and tech debt is a recipe for overcomplicating your future code maintenance. Creating a

modular source code design with clear, simple, and well-tested functions makes code maintenance easier. Clean and modular code leverages existing libraries, prioritizing readability and maintainability over micro-optimizations. These widely known practices simplify maintenance, reducing the time and effort needed for debugging and updates.

- **Streamlining Deployment Processes:** Ad-hoc manual processes are a frequent source of unnecessary complexity in software engineering. Teams implement Continuous Integration/Continuous Deployment (CI/CD) pipelines using tools like Jenkins, GitHub Actions, or GitLab CI. These pipelines automate testing, building, and deployment processes, significantly reducing the potential for human error. This not only expedites the deployment process but also makes it more reliable, ensuring a smoother and more efficient workflow.
- **Simplifying User Interface (UI) Design With Standard Frameworks:** Using UI frameworks like Material Design to create consistent and straightforward interfaces significantly enhances the user experience while simplifying development. A standard UI reduces the cognitive load on developers and end-users and makes the application more straightforward to maintain, also leading to higher user satisfaction.

### **21.2.2: 2. ENJOY: What If This Could Be Fun?**

Combining essential tasks with pleasurable activities can enhance your productivity while maintaining a sense of well-being. Accept that it is possible and beneficial to **integrate work and play**. Pair the most essential activities with the most enjoyable ones. Embrace the idea that work and play can co-exist harmoniously. Transform tedious tasks into meaningful rituals, infusing them with purpose and enjoyment. Allow laughter and fun to lighten more of your moments, turning routine activities into opportunities for joy and creativity.

This approach helps you stay engaged and motivated, making even the most mundane tasks feel more fulfilling and less burdensome. Letting joy and laughter permeate your daily routine creates a positive, dynamic environment where productivity and happiness thrive together.



image by azmanl from istock

Pairing essential activities with enjoyable ones creates a harmonious and engaging work environment for IT and software engineering professionals. This approach enhances productivity, promotes well-being, and increases job satisfaction by turning routine tasks into opportunities for joy and creativity.

Here are a few examples that IT architects can use as inspiration to help their organizations integrate work and play:

- **Architecture Reviews as Collaborative Learning Sessions:** Developers often see architecture reviews as bureaucratic, tedious, and time-consuming, but you can transform them into collaborative learning sessions. Adding a gamification element, such as rewarding the most insightful feedback, can make such reviews social and enjoyable. This engagement leads to better designs and stronger team cohesion.
- **Writing Technical Documentation:** Technical documentation, often perceived as monotonous, can be made more enjoyable by infusing creativity. Using storytelling techniques or incorporating visual elements like diagrams, infographics, and comic strips can make the task engaging. Encouraging personal anecdotes or humor where appropriate results in more comprehensive and user-friendly

documentation, making the process enjoyable for both writers and readers.

- **Keeping Up with New Technologies:** Continuous learning and keeping up with new technologies can feel overwhelming and exhausting. Aligning learning activities with personal interests and hobbies helps maintain motivation and enthusiasm. For instance, exploring game development or gamification techniques for team members who enjoy gaming, or delving into UI/UX design trends for those who love graphic design, makes continuous learning more enjoyable and fulfilling.

### 21.2.3: 3. RELEASE: The Power Of Letting Go

As the saying goes, “***the best thing one can do when it is raining is to let it rain.***” By acknowledging and embracing our circumstances, we can focus on what we have rather than what we lack, fostering gratitude and emotional resilience.

Regrets that continue to haunt us, grudges we can’t seem to let go of, and expectations that once were realistic but now hinder us all contribute to our emotional burdens. When we fall victim to misfortune, we easily obsess, lament, or complain about all we have lost. **Complaining** is one of the easiest things to do.



image by mixmike from istock

Similarly, IT and software engineering professionals can release unnecessary **emotional burdens**, fostering a positive and resilient mindset that enhances personal well-being and professional effectiveness.

Here are a few examples that IT architects as an inspiration to help their organizations let go:

- **Overcoming Regret Over Past Decisions:** Engineers and IT architects may regret choosing a particular technology stack or making architectural decisions that later proved suboptimal. Accept that decisions were likely made with the best knowledge available at the time and focus on learning from past experiences rather than dwelling on them. When regret surfaces, reflect on a positive outcome or lesson learned from that decision and express gratitude for the growth it provided. Shifting focus from regret to learning fosters a growth mindset and empowers engineers to make informed decisions without being weighed down by past mistakes.
- **Letting Go of Grudges Against Team Members:** Holding grudges against colleagues for past misunderstandings or conflicts can create a toxic work environment. Address and resolve conflicts through open communication and empathy, recognizing that holding grudges serves no constructive purpose. Each time a grudge resurfaces, consciously let it go by acknowledging the colleague's positive trait or contribution. Releasing grudges promotes a harmonious and collaborative team environment, improving morale and productivity.
- **Embracing Change and Letting Go of Resistance:** Resistance to adopting new technologies or methodologies can hinder progress and innovation. Be open to change and willing to experiment with the latest tools and approaches, viewing change as an opportunity for growth rather than a threat. When encountering resistance to change, identify one positive aspect or potential benefit of the new approach and express gratitude for the opportunity to innovate. Embracing change fosters innovation and keeps the team adaptable and forward-thinking, driving continuous improvement and success.

### 21.2.4: 4. REST: Take a Break

By incorporating periods of **rest and reflection**, you create a balanced routine that enhances productivity and overall quality of life. This structured approach allows you to maintain high levels of energy and concentration, preventing burnout and ensuring that you can consistently perform at your best. Embrace the art of doing nothing. Avoid overextending yourself by not doing more today than you can fully recover from by tomorrow. This approach promotes sustainable productivity and well-being.



image by chinnapong from istock

By embracing the art of doing nothing and incorporating structured work sessions, rest periods, and balanced routines, IT and software engineering professionals can enhance their productivity and well-being. This approach promotes sustainable productivity, prevents burnout, and ensures that engineers can consistently perform at their best.

Here are a few examples that IT architects can use as inspiration to help their organizations take a break:

- **Incorporating Rest and Reflection Periods:** Engineers frequently move from one task to another without taking time to reflect and rest, leading to mental fatigue. Scheduling short breaks after each

focused session for rest and reflection can mitigate this. Activities such as walking, meditating, or simply quietly sitting can be beneficial. For instance, after each 90-minute session, a 15-minute walk outside or 10 minutes of meditation can help clear the mind. These regular breaks help maintain mental clarity and focus, enhancing overall productivity and well-being.

- **Balancing Workload and Recovery:** Taking on more work than one can recover overnight leads to cumulative fatigue and stress for engineers. Effectively managing the workload ensures daily tasks are balanced and realistic, avoiding overextension. Tools like task lists and time-blocking can help manage workload effectively. Prioritizing tasks and allocating time based on their complexity and importance while avoiding scheduling back-to-back high-intensity tasks ensures recovery and consistent performance, preventing long-term fatigue.
- **Creating a Balanced Routine with Leisure Activities:** Continuous work without leisure can drain motivation and creativity. Incorporating leisure activities into the daily routine provides a mental break and stimulates creativity. Dedicating time in the evening for hobbies, social activities, or simply relaxing with a book or movie can be refreshing. Balancing work with leisure activities promotes overall well-being and helps maintain motivation and creativity.

### **21.2.5: 5. NOTICE: How to See Clearly**

Too often, we are physically with people but **not mentally present**. We struggle to notice them and see them truly. Being fully present makes others feel like the most important person in the world, even for a brief moment. This experience of undivided attention has a magical power that can leave a lasting impact long after the moment has passed.

People often describe the feeling of being with someone who is fully present as if that person had moved mountains for them. The power of presence is not about grand gestures but about being **wholly attentive** and **engaged** in the moment. This profound presence can transform relationships and create meaningful connections that resonate deeply.

Harness the power of presence to achieve a state of heightened awareness. Train your brain to focus on what is essential and ignore the irrelevant.

This practice improves your productivity and enriches your interactions with others.

Set aside your opinions, advice, and judgment to truly see others. Prioritize their truth above your own. Clear the clutter in your physical environment before tackling the clutter in your mind. This process of decluttering helps create a space conducive to presence.



image by portra from istock

IT and software engineering professionals can achieve a state of heightened awareness, improving productivity and enriching their interactions with others. This approach fosters a work environment where presence lead to meaningful connections and effective collaboration.

Here are a few examples that IT architects as an inspiration to help their organizations see clearly:

- **Truly Seeing and Listening to Colleagues:** Meetings and discussions often involve multitasking, where participants check emails or think about other tasks, leading to ineffective communication. Active listening is essential to fully engaging with colleagues during meetings and one-on-one interactions. During a meeting, put away your phone and close unnecessary tabs on your computer. Make eye contact, listen actively, and acknowledge the speaker's points before

responding. This engagement ensures that colleagues feel valued and heard, improving team dynamics and fostering a collaborative work environment.

- **Decluttering Physical and Digital Workspaces:** A cluttered workspace can lead to a cluttered mind, reducing efficiency and increasing stress. Clearing physical and digital clutter helps create an organized and focused work environment. An organized workspace reduces distractions, helping you maintain focus and clarity throughout the day.
- **Prioritizing Important Tasks Over Urgent Ones:** Engineers often get caught up in urgent but less important tasks, neglecting critical long-term projects. Using a prioritization matrix like the Eisenhower Matrix helps distinguish between important and urgent tasks. Focusing on essential tasks enhances long-term productivity and progress, preventing the constant firefighting of urgent but less impactful tasks.

## 21.3: Effortless Action

When you reach an Effortless State, you can perform Effortless Actions. Effortless Action is the art of **accomplishing more by trying less**. It involves finding a natural flow in your tasks and responsibilities, allowing you to achieve your goals with minimal strain. The process begins with taking the first obvious step, which helps overcome procrastination and sets you in motion. By avoiding overthinking, you can focus on reaching completion without getting bogged down in unnecessary details, preventing mental fatigue and keeping you moving forward (Figure 3).

Progress in Effortless Action is made by **pacing yourself** rather than **powering through**. This sustainable approach ensures a steady momentum without the risk of burnout. Effortless Action allows you to exceed expectations without excessive effort, enabling you to overachieve while preserving your energy and well-being. Ultimately, it's about aligning your efforts with a natural rhythm, making work feel less like a struggle and more like a seamless part of your day.



*Figure 3: Effortless Action is a part of the McKeown's Effortless framework (State, Action, Results).*

### 21.3.1: 6. DEFINE: What “Done” Looks Like

To begin an important project effectively, start by defining what “done” looks like. Establish **clear conditions for completion**, outlining specific criteria that indicate the project is finished. This clarity **helps you**

**stay focused** and prevents **unnecessary overextension**. Once these conditions are met, stop and acknowledge your progress.

McKeown suggests some simple techniques that everyone can use daily to focus and have a stronger sense of establishment. For instance, create a “**Done for the Day**” list, limited to items that would constitute meaningful progress. This list should include achievable tasks that contribute significantly toward your overall goal. Focusing on these critical activities ensures that each day ends with a sense of accomplishment and momentum.



image by evgenyatamanenko from istock

Similarly, by defining what “done” looks like, IT and software engineering professionals can stay focused on their goals, prevent overextension, and maintain a clear direction in their work. This approach fosters productivity, ensures high-quality outcomes, and provides a satisfying sense of accomplishment.

Here are a few examples that IT architects can use as inspiration to help their organizations better define what “done” looks like:

- **Completing a Feature Development:** Developing a new feature for a software application can often lead to scope creep without clear boundaries. To address this, defining what constitutes “done” for the feature is essential, as well as specifying criteria such as

passing all unit and integration tests, undergoing code review, and updating documentation. By documenting these conditions—“Feature X is done when it passes all unit tests, integrates without errors, is reviewed by a peer, and is documented in the user manual”—the development team can stay focused, avoid unnecessary additions, and ensure timely completion.

- **Finishing a Bug Fix:** Bug fixes can create a cycle of finding new issues without a clear endpoint. To prevent this, determine the conditions under which the bug fix is complete. These conditions include reproducing the problem, implementing the fix, testing it in different environments, and updating the issue tracker. Document the steps required to complete the bug fix: “Bug Z is done when the issue is reproduced, fixed, tested in staging and production environments, and marked as resolved in the issue tracker.” Explicit criteria for “done” help developers avoid endless bug-fixing cycles and ensure fixes are properly verified and documented.
- **Completing a Code Review:** Code reviews can drag on indefinitely without clear criteria for completion. To streamline this process, set clear criteria for a complete code review, such as checking for adherence to coding standards, verifying functionality, and ensuring no critical issues remain. Outline the steps: “The code review is done when the code adheres to our standards, all tests pass, and any identified issues have been addressed or noted for future improvement.” Clear completion criteria make code reviews more efficient and ensure they add value without becoming a bottleneck.

### 21.3.2: 7. START: The First Obvious Action

“Done” not only helps finish a project, but it also helps start it. Establishing clear conditions for completion and outlining specific criteria that indicate the project is finished enables you to stay focused, prevents unnecessary overextension, and provides a clear starting point.

But “done” is not enough. People still get stuck because they do not know how to start. Make the **first action the most obvious one**. Break this initial action into the tiniest, most concrete step possible, and then name it. For example, if the project is to write a report, the first step could be as simple as “open a new document.”

McKeown suggests some simple techniques to identify the first obvious action:

- Spend **sixty seconds** focusing on your desired outcome. Visualize what success looks like, and remember this image as you work. This brief period of concentration aligns your efforts and provides clear direction, making it easier to start.
- Gain maximum learning from a minimal viable effort by starting with a **ten-minute microburst** of focused activity. This short, intense work period can boost motivation and energy, making diving deeper into the project more manageable. This approach helps you overcome the inertia of starting and builds momentum for continued progress.



image by christianchan from istock

By breaking down the first action into the most apparent, tiny steps and visualizing the desired outcome, IT and software engineering professionals can effectively begin essential projects, build momentum, and ensure clear direction from the start. This approach reduces inertia and makes it easier to dive deeper into tasks confidently.

Here are a few examples that IT architects can use as inspiration to help their organizations find first obvious actions:

- **Beginning a New Feature Development:** In developing a new feature for a software application, the feature is considered complete when it passes unit tests, is integrated into the main codebase, and is documented. The first obvious action is to create a new branch in the version control system. This action involves opening the version control tool (e.g., Git) and executing the command to create a new branch: `git checkout -b new-feature-branch`. This small, concrete action provides a clear starting point and sets up the environment for feature development.
- **Initiating a Bug Fix:** When fixing a critical bug in the software, the bug fix is considered complete when the issue is reproduced, fixed, tested, and verified in the staging environment. The first obvious action is to reproduce the bug in the development environment. This action involves identifying the conditions under which the bug occurs and replicating those conditions in your development setup. Successfully reproducing the bug provides a clear starting point for identifying the cause and developing a fix.
- **Initiating a Documentation Task:** In updating the user manual with new feature details, the documentation update is complete when all new features are described with examples and integrated into the manual. The first obvious action is opening the text editor's documentation file. Locate the user manual file and open it using your preferred text editor (e.g., MS Word, Google Docs). Opening the document, while trivial, is an essential step in creating a starting point for adding new information.

### 21.3.3: 8. SIMPLIFY: Start With Zero

To simplify the process of completing an essential project, don't focus on simplifying the steps; instead, **remove unnecessary steps** altogether. Recognize that not everything requires going the extra mile. Minimizing the actions you need to take will streamline your workflow and conserve energy. Maximize the steps not taken and measure progress in the smallest increments to ensure steady advancement.

This concept aligns with the “Swedish Death Cleaning philosophy,” which involves decluttering your life by eliminating accumulated unnecessary items. Apply this principle to your project by removing redundant tasks and focusing only on what truly matters. This approach helps you

maintain clarity and efficiency, ensuring that your efforts are directed toward meaningful progress without being bogged down by extraneous activities.



image by arismart from istock

By applying the “Start With Zero” approach, IT and software engineering professionals can eliminate unnecessary steps, streamline workflows, and focus on what truly matters. This approach enhances productivity and ensures that efforts are directed toward meaningful progress, making projects more efficient and manageable.

Here are a few examples that IT architects can use as inspiration to help their organizations start with zero:

- **Streamlining Feature Development:** In developing a new feature for a software application, the traditional approach typically involves extensive planning, multiple design iterations, and comprehensive testing phases. The simplified approach starts with zero by eliminating unnecessary steps like excessive design iterations and over-engineering. Implementation begins with writing a simple prototype or MVP (Minimum Viable Product) that includes only the core functionality, focusing on delivering the essential feature first. By removing unnecessary steps and focusing on the core functionality, the feature is developed more quickly and can be tested and iterated based on user feedback.

- **Reducing Meetings:** The traditional approach for a collaborative team project involves frequent status meetings, detailed progress reports, and extensive planning sessions. The simplified approach starts with zero by eliminating unnecessary meetings and excessive reporting. Frequent meetings are replaced with asynchronous updates using collaboration tools like Slack or Trello, and only essential meetings with clear agendas and time limits are held. Reducing unnecessary meetings frees up time for actual work, increasing productivity and maintaining focus on important tasks.
- **Optimizing Deployment Processes:** The traditional approach to deploying a new software application version involves multiple manual steps, extensive testing environments, and comprehensive deployment checklists. The simplified approach starts with zero by removing redundant manual steps and streamlining the process. The implementation uses Continuous Integration/Continuous Deployment (CI/CD) pipelines to automate the deployment process, ensuring automated tests are in place to catch issues early. Automating the deployment process reduces human error, speeds up releases, and provides consistent quality.

#### **21.3.4: 9. PROGRESS: The Courage to Be Rubbish**

When beginning a project, it is crucial to adopt the mindset that it is perfectly acceptable to **start with less-than-perfect work**. Embracing a “zero-draft” approach, simply putting any words on the page, can be incredibly liberating. This technique effectively bypasses the paralysis often caused by perfectionism, allowing creativity to flow more freely. Accepting the idea of failing cheaply and making small and manageable mistakes early in the process accelerates learning and enhances decision-making skills over time.

Fred Brooks encapsulated this wisdom: *“Good judgment comes from experience, and experience comes from bad judgment.”* This quote highlights the necessity of mistakes for achieving mastery. Initial drafts are not meant to be perfect. Courage to produce imperfect work is a vital component of growth. By starting messy and allowing for errors, a solid foundation for continuous improvement and eventual mastery is established.



image by cristian gheorghe from istock

By embracing the courage to be rubbish and starting with imperfect versions, IT and software engineering professionals can bypass perfectionism, accelerate learning, and create a foundation for continuous improvement and eventual mastery. This approach encourages experimentation and iteration, leading to better outcomes over time.

Here are a few examples that IT architects can use as inspiration to help their organizations have the courage to create rubbish:

- **Starting a New Feature:** In developing a new feature for a software application, adopt the mindset that the first version might be rudimentary and full of flaws. Begin with zero-draft code, writing the initial version of the feature without worrying about perfection. Focus on getting a basic version that works, even if it's not optimized or clean. Start by coding the main functionality with simple logic, ignoring optimizations for now. This "rubbish" version allows you to quickly identify the main challenges and requirements, setting a foundation for iterative improvements.
- **Initial Project Planning:** When planning a new software development project, accept that the first project plan will be incomplete and potentially unrealistic. Begin with a zero-draft plan, drafting a rough outline of the project timeline, key milestones, and resource

allocations without striving for perfect accuracy. Create a simple Gantt chart or list of milestones using tools like Trello or a whiteboard. This initial plan provides a starting point for discussion and refinement, allowing the team to identify potential issues and adjust accordingly.

- **Learning a New Technology:** When learning a new programming language or framework, accept that your first attempts will be full of mistakes and inefficiencies. Begin with zero-draft learning, writing simple programs or small projects to get a feel for the syntax and features without worrying about best practices. Follow beginner tutorials and write code to replicate examples, making mistakes. These initial attempts build familiarity with the new technology and provide a foundation for more advanced learning and application.

### 21.3.5: 10. PACE: Slow if Smooth, Smooth is Fast

Maintaining a deliberate and **measured** pace can achieve more meaningful and lasting results without exhausting yourself. This balanced approach allows you to work efficiently while preserving your well-being and maintaining a high output standard.

To achieve sustained productivity, set an effortless pace: slow is smooth, smooth is fast. Reject the **false economy** of “powering through,” which often leads to burnout and decreased efficiency. Instead, create a balanced approach to your work by defining a suitable range for your efforts: determine that you will never do less than X and never more than Y. This ensures a consistent, manageable workload that promotes steady progress. Recognize that not all progress is created equal. Focus on the quality and significance of your achievements rather than merely the quantity.



image by flashvector from istock

By adopting a balanced approach and setting a sustainable pace, IT and software engineering professionals can maintain high productivity while ensuring quality and preserving their well-being. This method emphasizes the importance of consistent, manageable efforts that lead to meaningful and lasting results.

Here are a few examples that IT architects can use as inspiration to help their organizations set an effortless pace:

- **Learning a New Technology:** When learning a new programming language or framework, the traditional approach might involve intensive, uninterrupted study sessions that lead to quick burnout and poor retention. A balanced approach involves studying for 30 minutes and no more than 2 hours per day. Integrate learning into daily routines with consistent, shorter study sessions, allowing time for reflection and practice. Steady, consistent learning leads to better understanding and retention of new skills.
- **Managing Code Development:** The traditional approach to developing a complex feature for a software application often involves powering through long hours of coding without breaks, leading to burnout and errors. A balanced approach involves determining that you will code for no less than 4 hours and 6 hours a day, with breaks in between. Implement this by scheduling coding sessions with regular short breaks. This steady pace prevents burnout, reduces mistakes, and ensures high-quality code over time.
- **Debugging and Testing:** The traditional approach to debugging and testing a new feature often involves marathon sessions that lead

to frustration and oversight. A balanced approach allocates no less than 1 hour and 3 hours daily for debugging and testing. Approach debugging methodically with regular breaks to reassess and strategize. This measured pace allows for more effective problem-solving and thorough testing, leading to more robust software.

## 21.4: Effortless Results

Effortless Results are the natural outcome of consistently cultivating your Effortless State and taking Effortless Action. By maintaining a clear objective, breaking tasks into tiny, obvious first steps, and working at a consistent, manageable pace, you achieve your desired outcomes with greater ease. However, the true power of Effortless Results lies in their sustainability (Figure 4).

Effortless Results are those that continue to **flow to you repeatedly**, with minimal additional effort. You've established a system where **success becomes a cycle** rather than a one-time event. By refining your process and eliminating unnecessary steps, you ensure that your efforts yield ongoing benefits. This approach allows you to maintain high productivity and achieve your goals consistently, creating a seamless and continuous flow of accomplishments.

In essence, Effortless Results are about creating a **self-sustaining loop** of success. By leveraging the principles of the Effortless State and Effortless Action, you set the stage for ongoing achievement, making it possible to reach your objectives repeatedly without constant exertion.



**Figure 4:** *Effortless Results is a part of the McKeown's Effortless framework (State, Action, Results).*

### **21.4.1: 11. LEARN: Leverage the Best of What Others Know**

To achieve Effortless Results, McKeown argues, it is crucial to **stand on the shoulders of giants**. That is, to leverage the best of what experts and pioneers have discovered. Use their knowledge as a foundation to build upon, enabling you to achieve more with less effort.

In addition, one must focus on **learning principles**, not just facts and methods. Understanding first principles allows you to apply them repeatedly across various contexts. You can quickly adapt and innovate by grounding yourself in these fundamental truths, making complex tasks more straightforward and manageable.

Effortless Results stem from a **deep understanding of first principles** and the ability to apply them creatively and consistently. By building on the knowledge of others and developing your unique insights, you create a sustainable pathway to continuous achievement and innovation.



image by lvcandy from istock

By understanding and applying first principles, leveraging experts' knowledge, and building unique insights, IT and software engineering professionals can achieve effortless results. This approach fosters innovation and continuous improvement, ensuring that complex tasks become more manageable and maximize productive efforts.

Here are a few examples that IT architects can use as inspiration to help their organizations leverage the best of what others know:

- **Applying Design Patterns in Software Development:** When developing a scalable and maintainable software application, start with the first principles by learning the fundamentals behind common design patterns such as Singleton, Factory, Observer, and MVC. Study books like "Design Patterns: Elements of Reusable Object-Oriented Software" by the Gang of Four. Apply these

patterns appropriately in the software architecture to address scalability and maintainability issues. Leveraging well-established design patterns allows for efficiently building a robust, proven, and flexible application framework.

- **Innovating with New Technologies:** To integrate machine learning into an existing product, start with the first principles by understanding the basics of machine learning algorithms, data preprocessing, model training, and evaluation. Use open-source libraries like TensorFlow or PyTorch and build on existing models and research to implement the solution. Using foundational knowledge and the work of experts facilitates the seamless integration of advanced technologies, enhancing functionality with minimal effort.
- **Optimizing Database Performance:** To optimize the performance of a relational database, start with first principles by learning the fundamentals of database normalization, indexing, query optimization, and transaction management. Study best practices and methodologies from experts. Apply indexing strategies, optimize queries based on execution plans, and configure database settings according to expert recommendations. Achieving optimal database performance through a deep understanding of underlying principles and expert advice leads to faster and more efficient data handling.

#### **21.4.2: 12. LIFT: Harness the Strength of Ten**

**Teaching others** is an accelerated way to learn. When you prepare to teach, you increase your engagement, focus more intently, listen to understand, and think about the underlying logic so you can articulate the ideas in your own words. This process reinforces your understanding and enhances your ability to convey complex concepts.

Use teaching as a lever to harness the strength of ten, achieving a far-reaching impact by teaching and by **teaching others to teach**. You'll notice how much you learn when you live what you teach. You can disseminate knowledge effectively by telling stories that are easy to understand and repeat.

Ensuring your messages are easy to understand and hard to misunderstand is not just a communication strategy. It's a powerful tool for **amplifying your impact**. By simplifying and clarifying your communication,

you make it easier for others to grasp and remember the knowledge you share, thereby increasing the reach and effectiveness of your message. This process underscores your crucial role in knowledge sharing and the impact you can have on others.

For instance, one motivation for writing this book is to create reusable material that teaches others about pragmatic approaches to running an architecture practice. It also gives them material they can reuse to teach others the same.



image by nattakorn maneerat from istock

By teaching and teaching others to teach, IT and software engineering professionals can amplify their impact and foster a culture of continuous learning and improvement. This approach reinforces their knowledge and equips the team with the skills and understanding necessary for greater efficiency and innovation.

Here are a few examples that IT architects can use as an inspiration to help their organizations harness the strength of ten:

- **Promoting Continuous Integration/Continuous Deployment (CI/CD):** An engineer who has successfully implemented CI/CD pipelines in past projects runs workshops to teach the team how to set up and maintain CI/CD pipelines. Providing hands-on experience with popular tools like Jenkins, GitHub Actions, or

GitLab CI, the engineer creates step-by-step tutorials and shares best practices. Using real examples from the current project to demonstrate the impact of CI/CD, the team learns to automate testing and deployment, leading to more reliable releases and a faster development cycle.

- **Enhancing Security Practices:** A security expert aims to improve the team's approach to cybersecurity by conducting security training sessions to educate team members on best practices, common vulnerabilities, and mitigation strategies. Using simple, memorable stories to illustrate the importance of security, the expert provides checklists and templates for secure coding and threat modeling. The outcome is that team members become more security-conscious and capable of implementing robust security measures, reducing the risk of breaches.
- **Mentoring on Effective Documentation:** A developer who excels at creating clear, concise documentation holds workshops to teach the team how to write effective documentation. Sessions cover different types of documentation, such as ADRs, RfCs, API documentation, user guides, and technical specs. By providing examples of well-written documentation and highlighting key elements that make it effective, the developer encourages team members to practice and peer-review each other's work. Improved documentation quality across the team leads to better knowledge sharing and easier onboarding for new team members.

### **21.4.3: 13. AUTOMATE: Do It Once and Never Again**

Automating as many essential tasks as possible frees up space in your brain. This space allows you to focus your mental energy on more important matters. One effective way to ensure consistency and accuracy is using checklists, which help you get it right every time without relying on memory. Automation creates a more efficient and less cluttered mental environment, enabling you to perform at your best with minimal effort.



image by shutter2u from istock

By automating essential tasks and using checklists for routine processes, IT and software engineering professionals can streamline their workflows, reduce errors, and free up mental space for more important activities. This approach leads to increased efficiency, consistency, and overall productivity.

Here are a few examples that IT architects can use as inspiration to help their organizations automate things:

- **Automating Code Quality Checks:** Ensuring code quality through manual reviews and testing can be time-consuming and inconsistent. A better approach involves high-tech automation: Integrate automated code quality tools like SonarQube or ESLint into your CI/CD pipeline. Configure the tools to run automated checks on every commit, providing immediate feedback on code quality issues. The outcome is consistent and automated code quality checks that improve overall code standards and reduce the need for manual reviews.
- **Standardizing Development Environments:** Setting up development environments manually for new team members or new projects can be time-consuming and error-prone. The approach involves high-tech automation by using containerization tools like Docker to create standardized development environments. Create Docker images that include all necessary tools, libraries, and con-

figurations, and share them with the team. The outcome is that new environments can be set up quickly and consistently, reducing setup time and avoiding configuration issues.

- **Automating Data Management:** Managing and updating databases manually can be prone to errors and inconsistencies. The approach involves high-tech automation by implementing automated data management scripts using tools like Python or SQL scripts scheduled with cron jobs or similar scheduling tools. Write scripts to handle routine data management tasks such as backups, updates, and migrations. The outcome is automated data management that ensures data integrity and frees up time for more strategic tasks.

#### **21.4.4: 14. TRUST: The Engine of High-Leverage Teams**

When trust exists in your relationships, they require less effort to maintain and manage. You can quickly and efficiently split work between team members. People feel comfortable **discussing problems openly and honestly**, sharing valuable information rather than hoarding it. This environment encourages team members to ask questions when they don't understand something. Consequently, the speed and quality of decisions improve, political infighting decreases, and you may even enjoy the experience of working together. This dynamic allows you to focus your energy and attention on getting important tasks done rather than on simply getting along.

Conversely, when trust is low, **everything becomes difficult**. Sending a simple text or email is exhausting as you weigh every word for how it might be perceived. Responses may induce anxiety, and every conversation feels like a grind. Without trust in someone's ability to deliver, you need to check up on them constantly, remind them of deadlines, hover over their work, or avoid delegating tasks altogether, believing it's easier to do it yourself. This lack of trust can cause work to stall and impede team performance.

Inside every team are individuals with interrelated roles and responsibilities, moving at high speeds. Without trust, conflicting goals, priorities, and agendas create friction and wear everyone down. **Trust acts like**

**engine oil**, lubricating the team's interactions and keeping them working smoothly together. A team running out of trust will likely stall or sputter out.

Every relationship involves three parties, **Person A** and **Person B**, and **the structure** that governs them. When trust becomes an issue, most people point fingers at the other person, be it the manager blaming the employee or vice versa. However, every relationship has a structure, even if it's an unspoken and unclear one. Unclear expectations, incompatible or conflicting goals, ambiguous roles and rules, and misaligned priorities and incentives characterize a low-trust structure.

High-trust agreements help mitigate these issues by clarifying:

- **Results:** What results do we want?
- **Roles:** Who is doing what?
- **Rules:** What minimum viable standards must be kept?
- **Resources:** What resources (people, money, tools) are available and needed?
- **Rewards:** How will Progress be evaluated and rewarded?

Establishing clear expectations and structures creates a high-trust environment that enables teams to function efficiently and effectively.

Leverage trust as the essential lubricant of frictionless and high-functioning teams. Making the right hire once can produce results repeatedly. Follow the Three I's Rule: hire people with integrity, intelligence, and initiative. Design high-trust agreements to clarify results, roles, rules, resources, and rewards.

Being an architect is much **easier in high-trust organizations**. In low-trust organizations, people frequently expect architects to be **police agents**. IT governance processes frequently associated with an architecture practice are used or misused to force bureaucratic controls as teams often do not trust each other.



image by nathaphat from istock

By establishing and maintaining a high-trust environment, IT and software engineering teams can work more efficiently and effectively. Trust reduces friction, enables **better decision-making**, and fosters a positive, collaborative culture where team members feel valued and empowered.

Here are a few examples that IT architects can use as inspiration to help their organizations create a high-trust environment:

- **Open Communication and Problem-Solving:** To address a critical bug in the software that could delay the release, encourage an environment where team members feel comfortable discussing problems openly. Regularly schedule team stand-ups and retrospectives where issues can be raised without fear of blame. Encourage a culture where questions are welcomed, and information is shared freely. The outcome is an effective collaboration among team members to identify and resolve the bug quickly, leveraging their collective knowledge and skills.
- **Delegating Tasks Efficiently:** When delegating a complex module development to a junior engineer or IT architect, trust their capability to handle the task while providing guidance and support. Communicate the task requirements and expected outcomes. Provide access to resources and be available for consultation, but allow the engineer autonomy to approach the problem. The result is that the junior engineer gains confidence and develops skills, while senior team members can focus on other critical tasks, improving overall team productivity.

- **Making the Right Hire:** When hiring a new software engineer for a critical project, the approach involves following the Three I's Rule: focus on candidates with integrity, intelligence, and initiative. Design interview questions and assessments that evaluate these traits, such as scenario-based questions to gauge problem-solving skills and ethical dilemmas to assess integrity. The outcome is hiring the right candidate who increases team efficiency and reduces the need for constant oversight, as they can be trusted to deliver high-quality work independently.

#### **21.4.5: 15. PREVENT: Solve the Problem Before It Happens**

Just as you can find small actions to make your life easier in the future, look for small actions that will prevent your life from becoming more complicated. Simple preventative measures (such as setting reminders, automating tasks, or creating checklists) can significantly reduce the likelihood of future problems. Focusing on these small, strategic actions ensures a smoother, more efficient path forward, allowing you to achieve more with less effort and stress.

In other words, don't just manage problems—solve them before they happen. Seek simple actions today that can prevent complications tomorrow. Finding opportunities to invest a small amount of effort now can eliminate recurring frustrations and streamline your future.

This proactive approach is the long tail of time management. When you invest your time in actions with a long tail, you reap the benefits over a long period. For example, spending two minutes to organize your workspace can save you countless hours of searching for items in the future. Catch mistakes before they occur by adopting a measure-twice, cut-once mentality.

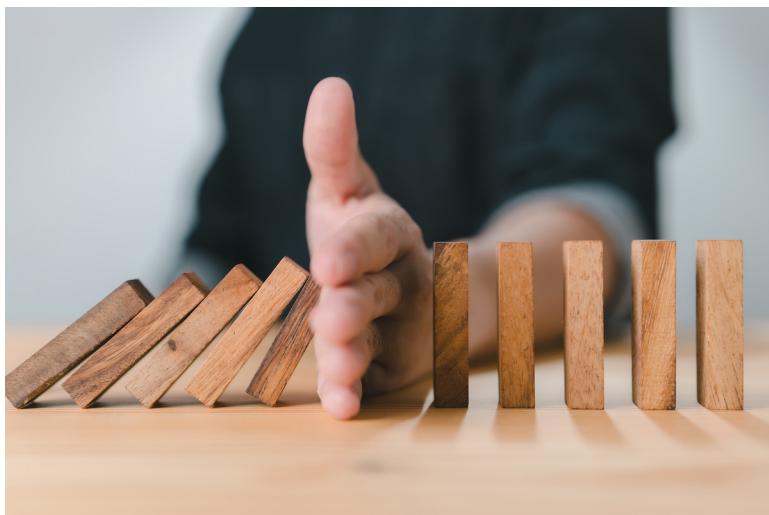


image by ipuwadol from istock

IT and software engineering teams can solve potential problems before they occur, leading to smoother operations, fewer disruptions, and a more efficient workflow. This proactive approach saves time and resources, allowing teams to focus on innovation and continuous improvement.

Here are a few examples that IT architects can use as inspiration to help their organizations prevent problems:

- **Regular Code Reviews:** Establishing a regular code review process can prevent inconsistent code quality and reduce the number of errors from unreviewed code changes. Make code reviews a mandatory part of the development workflow. Use pull request templates and review tools like GitHub or GitLab to ensure thorough reviews. The outcome is that regular code reviews maintain high code quality, catch potential issues early, and promote knowledge sharing among team members.
- **Setting Up Monitoring and Alerts:** Implementing monitoring and alerting systems can prevent the frequent discovery of system outages or performance issues after they impact users. Use tools like Prometheus, Grafana, and New Relic to monitor system performance and uptime. Set up alerts to notify the team of any anomalies or performance degradation. The outcome is early detection of

issues, allowing the team to address problems before they impact users, improving system reliability and user satisfaction.

- **Regular Security Audits:** Discovering security vulnerabilities after a breach can lead to significant downtime and data loss. You can reduce such problems with preventative actions, such as conducting regular security audits and implementing best practices. Schedule regular security audits using tools like OWASP ZAP and Nessus. Train the team on secure coding practices and conduct periodic security reviews. The outcome is that regular security audits and training reduce the risk of vulnerabilities, preventing potential breaches and ensuring data protection.

## 21.5: The Road to Effortless Achievement

The path to achieving great results doesn't have to be tortuous. Embracing the idea that things can be easy and even enjoyable opens you up to effortless solutions. By clearly defining your goal and identifying a small yet significant first step, you set yourself on a journey of effortless actions that lead to steady and meaningful progress.

When you leverage knowledge, automation, and trust, you create a system that produces recurring results. This system simplifies your life and enhances productivity, demonstrating that life doesn't have to be as hard and complicated as we often make it. Adopting a mindset that prioritizes ease and enjoyment enables a more streamlined and fulfilling approach to achieving your organizational goals.

## 21.6: To Probe Further

- Effortless: Make It Easier to Do What Matters Most<sup>4</sup>, by Greg McKeown, 2021
- Essentialism: The Disciplined Pursuit of Less<sup>5</sup>, by Greg McKeown, 2020
- Here's why enterprise IT is so complex<sup>6</sup>, by Gregor Hohpe, 2018
- Tidy First?<sup>7</sup>, by Kent Beck, 2023
- Execution Management Matrix<sup>8</sup>
- The Eisenhower Matrix<sup>9</sup>

---

<sup>4</sup><https://gregmckeown.com/books/effortless/>

<sup>5</sup><https://gregmckeown.com/books/essentialism/>

<sup>6</sup><https://architectelevator.com/architecture/it-complexity/>

<sup>7</sup><https://www.oreilly.com/library/view/tidy-first/9781098151232/>

<sup>8</sup><https://obren.io/tools/matrix/?matrix=executionManagement>

<sup>9</sup><https://obren.io/tools/matrix/?matrix=eisenhowerMatrix>

## 21.7: Questions to Consider

- *How does the idea of achieving goals with minimal strain resonate with your personal work ethic and habits?*
- *What tasks could benefit from automation to enhance your productivity and reduce unnecessary effort?*
- *Have you experienced moments when you were physically rested, emotionally unburdened, and mentally energized? What facilitated these moments?*
- *What mental clutter currently hinders your productivity, and how can you clear it to foster an effortless state?*
- *How can you apply the concept of “make the change easy, then make the easy change” in your work?*
- *How can you combine essential tasks with pleasurable activities to enhance your productivity and well-being?*
- *What regrets, grudges, or unrealistic expectations are you holding onto that might be hindering your progress?*
- *How do you currently balance work and rest to prevent burnout and maintain high performance?*
- *What structured routines can you implement to incorporate rest and reflection periods into your workday?*
- *What steps can you take to declutter your physical and digital workspaces to improve focus and efficiency?*
- *What experts and pioneers in your field can you learn from to build on their knowledge and achieve more with less effort?*
- *How can you leverage teaching as a tool to reinforce your knowledge and multiply your impact?*
- *What complex concepts can you simplify and clarify to make them easier for others to understand and remember?*
- *What clear expectations and structures can you establish to create a high-trust agreement in your projects?*
- *How can you adopt a proactive approach to identify and solve potential problems before they occur?*

# **Part IV: Expanding the Architect's Toolkit**

## **22: Expanding the Architect's Toolkit: Learning From Other Fields**



image by worawee meepian from istock

**IN THIS SECTION, YOU WILL:** Get a summary of several resources that I use as inspiration for running the Grounded Architecture practice in complex organizations.

A holistic approach is essential for achieving success in the intricate and fast-paced world of IT architecture, especially within complex organizations. Traditional IT architecture literature often fails to address the multifaceted challenges architects face today. To bridge this gap, I have curated a selection of resources that provide a broader perspective, drawing inspiration from social sciences, behavioral sciences, product management, and political sciences. These resources offer valuable insights and practical strategies for running the Grounded Architecture practice in complex organizational environments.

- **Architecture in Product-Led Organizations: Learning From Customer-Centric Fields:** Effective product development is a cornerstone of organizational success. Several essential resources for architects in this realm include Melissa Perri's "Escaping the Build Trap: How Effective Product Management Creates Real Value" and "The Discipline of Market Leaders" by Michael Treacy and Fred Wiersema. Perri's work highlights the pitfalls of focusing too narrowly on outputs rather than outcomes, urging architects to align product development with real customer value. Treacy and Wiersema provide a strategic framework for achieving market leadership by focusing on customer intimacy, product leadership, and operational excellence. These resources offer a comprehensive guide for architects to drive product-led growth and innovation.
- **Decision Intelligence in IT Architecture: Learning From Data, Social, and Managerial Fields:** Decision intelligence is a burgeoning field that combines data science, social science, and managerial science to transform information into actionable insights. In the context of IT architecture, decision intelligence is becoming increasingly vital. It equips architects with the tools and frameworks to analyze data effectively, foresee potential outcomes, and make strategic decisions that align with organizational goals. Understanding decision intelligence is essential for architects who aspire to drive innovation and efficiency in their projects.
- **Economic Modeling With ROI and Financial Options: Learning From the Finance Field:** Evaluating the economic impact of technology investments is a critical aspect of IT architecture. This section delves into two economic models that provide valuable insights into this evaluation process. The return on investment (ROI) metaphor helps architects assess the profitability and efficiency

of their investments. Meanwhile, the financial options metaphor offers a flexible approach to decision-making under uncertainty, allowing architects to consider various scenarios and outcomes. These models are instrumental in making informed financial decisions that support long-term organizational success.

- **How Big Transformations Get Done: Learning From Mega-Projects<sup>1</sup>:** IT transformation projects face similar challenges as other mega-projects, often failing due to cognitive biases and poor planning. However, applying key lessons from successful projects—such as risk mitigation, modular design, and stakeholder engagement—can significantly improve their outcomes.

By exploring these diverse resources, IT architects can significantly enhance their ability to drive strategic initiatives within their organizations. These insights provide a robust framework for developing a grounded and effective an architecture practice, enabling architects to address the unique challenges of their roles with confidence and precision. Whether you are an aspiring architect or a seasoned professional, these perspectives will offer valuable guidance on navigating and excelling in the dynamic field of IT architecture.

---

<sup>1</sup>[big-projects](#)

# **23: Architecture in Product-Led Organizations: Learning From Customer-Centric Fields**



image by tevarak from istock

**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-led organizations focus on building and delivering products that deliver value. 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 Management Creates Real Value<sup>1</sup>](https://www.goodreads.com/book/show/42611483-escaping-the-build-trap)” by Melissa Perri, and
- “[The Discipline of Market Leader<sup>2</sup>](https://hbr.org/1993/01/customer-intimacy-and-other-value-disciplines)” by Michael Treacy and Fred Wiersema.

---

<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>

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).

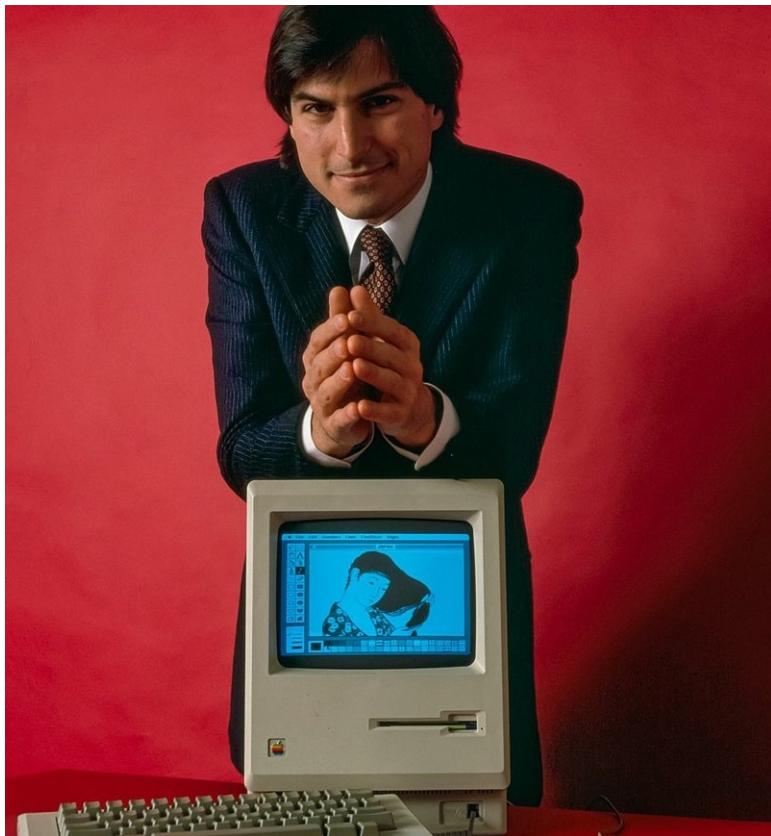


image by wikimedia commons

## 23.1: 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

### 23.1.1: 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.*

### 23.1.2: 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.

### **23.1.3: 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.

### 23.1.3.1: 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 khosrork from istock

### 23.1.3.2: 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.

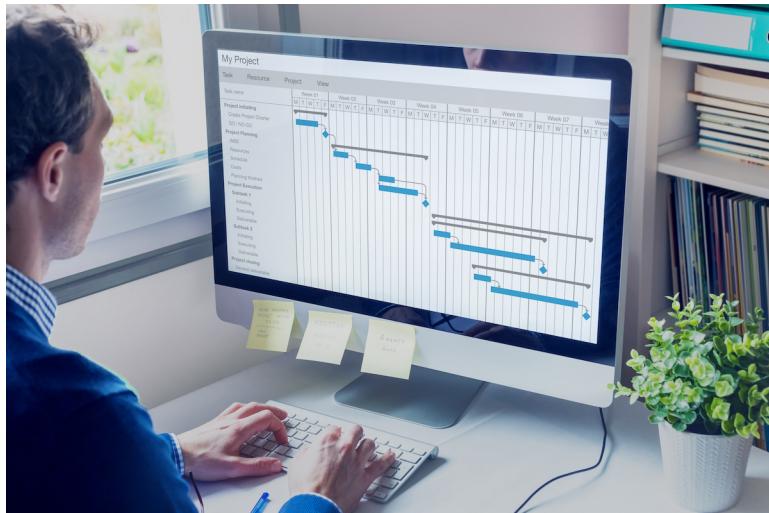


image by nicoelnino from istock

### 23.1.3.3: 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).



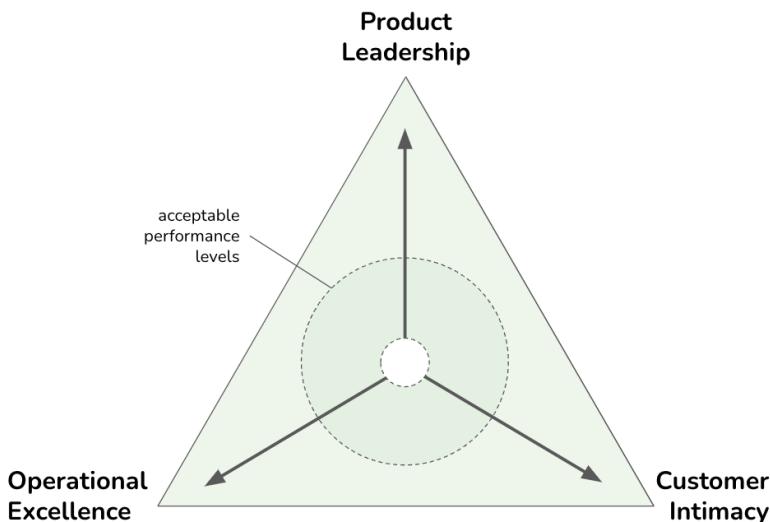
image by gorodenkoff from istock

---

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

## 23.2: 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 exploitation, and disjoint work procedures. Exemplars are Apple, Tesla, Nike, Rolex, and Harley-Davidson.

**Operational Excellence** companies provide reliable products and services

---

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

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.

## 23.3: Product Operations

Another product concept relevant to an architecture 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 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.

---

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

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.

### 23.3.1: 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**.

### **23.3.2: 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**.

### **23.3.3: 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.

### **23.3.4: Product Operations and Architecture Practice**

In many ways, the concept of Product Operations resonates with my view of an architecture 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, an architecture practice can **create powerful synergic relationships.** Like an 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 an 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. An architecture practice can help Product Operations with complementary insights, data, and stakeholder connections.

## 23.4: 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?*

## **24: Decision Intelligence in IT Architecture: Learning From Data, Social, and Managerial Fields**



image by sasun bughdaryan from istock

**IN THIS SECTION, YOU WILL:** Learn the basics of decision intelligence, the discipline of turning information into

better actions, and its relevance for an 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.
- Group decision-making offers significant advantages but increases complexity as it requires higher decision-making skills from each member.

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

---

<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/>

- 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.

Decision intelligence is the secret sauce of IT architects' work in every twist and turn.

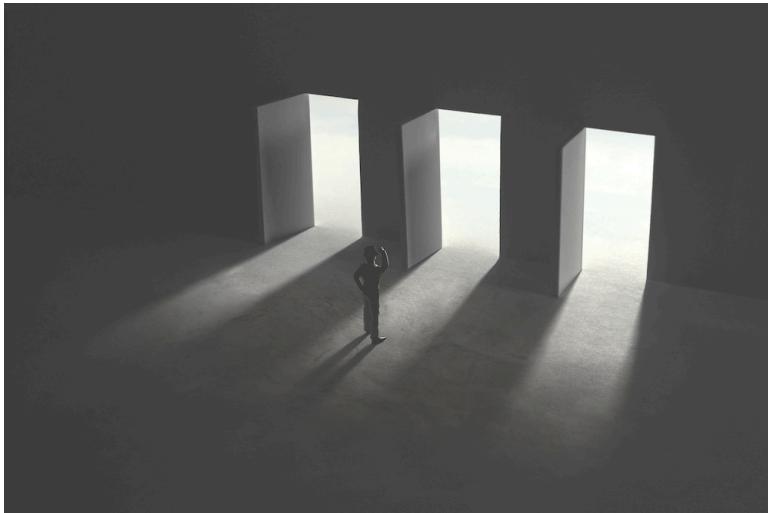


image by francescoch from istock

---

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

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

## 24.1: Basics of Decision-Making

Let's start with some basics: defining decisions, outcomes, and goals.

### 24.1.1: 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 in IT: vendor lock-in, which 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 proprietary technologies, high switching costs, contractual obligations, or the significant effort required to migrate data and systems.

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.

---

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

### 24.1.2: 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-making 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 terrible? 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>). On the flip side, however, COVID-19 boosted online tools' rocket boost. It birthed a new era of virtual collaboration (think Zoom, Microsoft Teams, Slack, Miro).

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

### 24.1.3: 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

---

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

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

determine 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—wholly unnecessary and probably 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 your decision's potential paradise and disaster scenarios. 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 resources—money, time, or effort—you'd spend for this flawless foresight.
3. **Balance Investment with Importance:** This little exercise helps you determine the value of achieving perfect clarity and making the best 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 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. This high-impact decision 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!

## 24.2: Preparing for Making Decisions

Decisions are the steering wheel for reaching our goals. Consequently, it is crucial to understand and define goals properly and to understand whether a decision needs to be made at all.

### 24.2.1: 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 run in circles. By identifying what’s truly important and ignoring everyone else’s distractions, you can focus on what really counts.

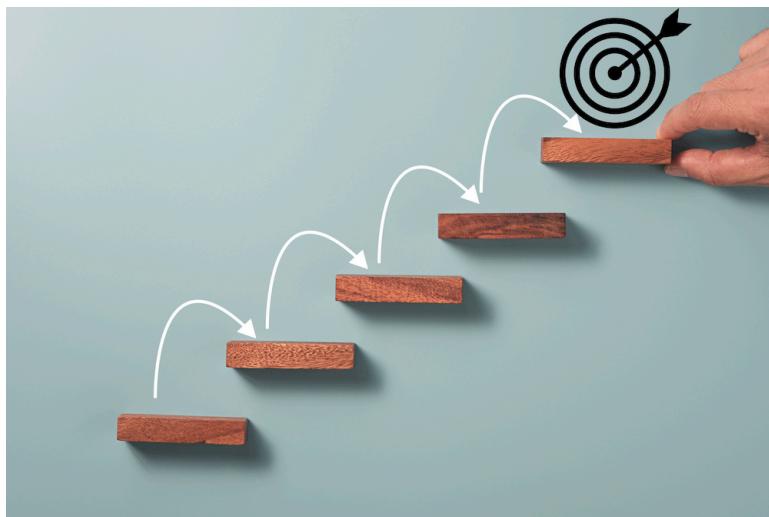


image by dilok klaisataporn from istock

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 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. It's 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, primarily within your control. In business, it's all about increasing website visits, slashing infrastructure costs, and boosting revenue. They're aspirational and tricky to manage, but hey, no pain, no gain.
- **Process Goals:** They're measurable and entirely 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 should always serve your higher aspirations.

You need all three types of goals neatly connected like a well-oiled machine. For example, running a flashy 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.

### **24.2.2: 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 about growth and expansion. At the same time, the managers (agents) might dream of longer lunch breaks and fatter paychecks. This **clash of interests** can lead to a mismanagement mess if mishandled.

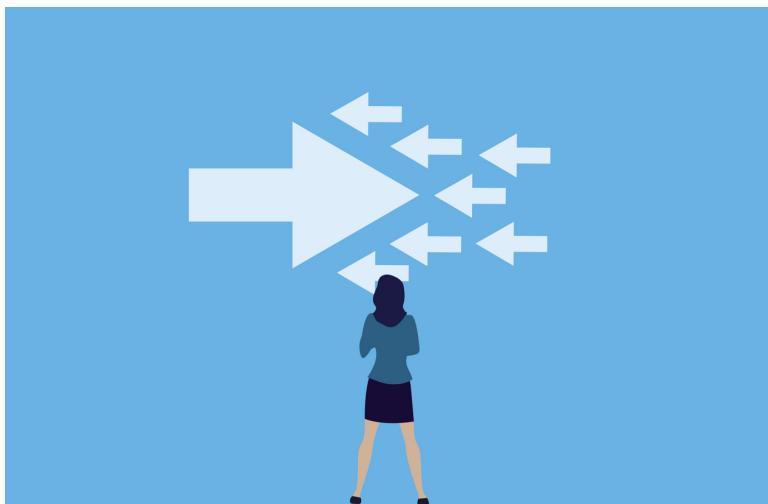


image by maria stavreva from istock

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 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,

---

<sup>9</sup><https://engineering.spotify.com/2020/08/how-we-use-golden-paths-to-solve-fragmentation-in-our-software-ecosystem/>

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.

### 24.2.3: 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 space for a decision to be made. Sometimes, the big cheese has already decided and needs you to rubber-stamp it like a bureaucratic formality.

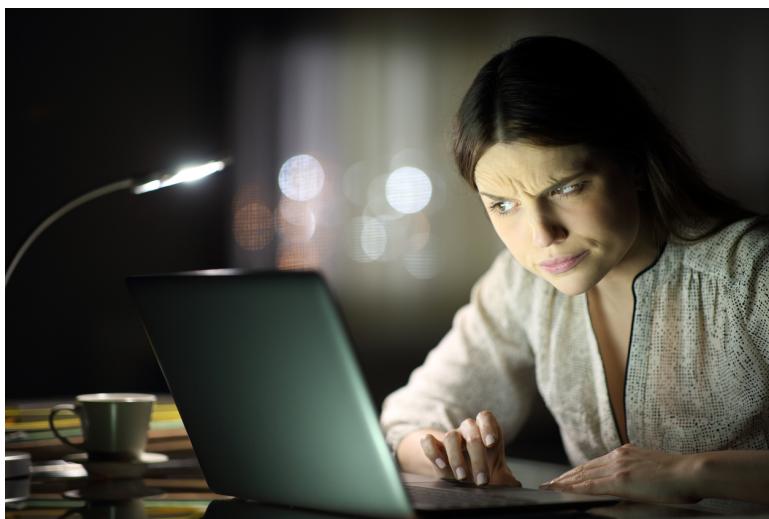


image by pheelings media from istock

Before diving into the murky waters of faux decision-making, clarify whether there's room for a decision. According to 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.
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.

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 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 a cameo appearance.

## 24.3: Decision-Making Complexity

Some decisions are easier to make than others. Kozyrkov identifies 14 factors that can make decision-making more or less complex:



image by chaiyapruek2520 from istock

### 24.3.1: 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.

### 24.3.2: 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. Still, it 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.

### **24.3.3: 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.

### **24.3.4: 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.

### **24.3.5: Costs of reversing a decision**

While decisions typically involve a commitment of resources you can't undo, some 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.

### **24.3.6: 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 IT architects' work involves creating visualizations and abstractions that can reduce cognitive load and help others make better decisions about complex systems.

### **24.3.7: 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.

### **24.3.8: Pressure and stress**

Decisions made under conditions of **low pressure and stress** are generally easier. In contrast, those made in high-pressure, stressful situations are more challenging.

### **24.3.9: 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.

### **24.3.10: 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. On the other hand, risk involves taking a known gamble, where you understand the potential consequences and likelihoods.

### **24.3.11: 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.

### **24.3.12: Impact on others**

Making decisions alone is generally easier. The process is simpler when you're the sole decision-maker, without involving others, and the decision's outcome impacts only you. 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.

### **24.3.13: 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.

### **24.3.14: 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.

## 24.4: Decision is a Step in the Process

The Radical Candor book introduces a [productivity-focused strategy<sup>10</sup>](#) to improve decision-making processes, reduce time spent in unproductive meetings, and enhance overall efficiency. The central premise of this methodology is that you cannot just make a decision. To have any impact, you need to go through several steps: Listen, Clarify, Debate, Decide, Persuade, Implement, and Learn.



image by t\_kimura from istock

### 24.4.1: Listen

The first step is to **Listen**. Gathering relevant data, feedback, and insights from various sources is crucial. Understanding different perspectives from stakeholders, team members, and experts helps form a comprehensive view of the situation. This step ensures a thorough **understanding of the situation** by gathering relevant data and insights from various sources. It

---

<sup>10</sup><https://www.radicalcandor.com/wp-content/uploads/2022/09/How-To-Get-Shit-Done-With-Radical-Candor.pdf>

also ensures consideration of **diverse perspectives**, which helps form a well-rounded view of the issue.

### **24.4.2: Clarify**

Next, you need to **Clarify**. Clearly articulate the issue or decision to ensure **everyone understands the context**. Establishing clear, specific goals and desired outcomes for the decision is essential. This sets the stage for focused discussions and aligned efforts. Clarifying the problem or decision you need to make ensures clear articulation and alignment on objectives and desired outcomes. This step **eliminates misunderstandings** and ambiguities, solidifying the foundation for focused discussions.

### **24.4.3: Debate**

Following clarification, the step is to **Debate**. Engage in discussing potential solutions and alternatives, weighing the **pros and cons** of each option. Encourage an **open discussion** where team members can freely share their ideas and constructively challenge each other's viewpoints. This collaborative approach helps in exploring various aspects of the decision. Debating potential solutions ensures a thorough evaluation of all possible options. It promotes an open exchange of ideas and constructive challenges, allowing for exploring various aspects and implications of the decision.

### **24.4.4: Decide**

Once the debate has provided a thorough evaluation of options, it's time to **Decide**. Choose the best course of action based on the gathered information and discussions. Depending on your organizational culture, you may reach a consensus among the team or have the decision made by a designated leader. The key is to make an **explicit and informed choice**. The decision-making step ensures an informed and thoughtful choice based on the available information and discussions. It provides a clear resolution and direction for moving forward, ensuring the best action is selected.

#### **24.4.5: Persuade**

After making the decision, the next step is to **Persuade**. Clearly **communicate** the decision and the rationale behind it to all stakeholders. It's essential to **gain buy-in** from team members and stakeholders by addressing their concerns and objections, convincing them of the decision's validity and importance. Persuading stakeholders and team members ensures effective communication of the decision and its rationale. This step secures buy-in and support, addressing and mitigating any concerns or objections that may arise.

#### **24.4.6: Implement**

With buy-in secured, you move to **Implement**. Execute the plan by **assigning tasks and responsibilities** to the appropriate individuals. Ensure everyone understands their role in the overall plan, providing the **necessary resources** and support to facilitate effective implementation. Alignment and clear communication are vital during this phase. Implementing the decision ensures the execution of the plan through the assignment of tasks and responsibilities. It ensures alignment and understanding of roles and the provision of necessary resources and support for successful implementation.

#### **24.4.7: Learn**

Finally, it's crucial to **Learn**. Monitor the implementation process and measure the outcomes against the set objectives. **Collect feedback** from stakeholders and team members to assess the effectiveness of the decision. **Analyze the results** to identify lessons learned and make necessary adjustments to improve future decision-making processes. This continuous improvement approach helps **refine strategies** and **enhance productivity** over time. Learning from the process ensures continuous improvement. Monitoring and measuring the implementation and outcomes, collecting and analyzing stakeholder feedback, and identifying lessons learned all refine future decision-making processes.

## 24.5: 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 [Lightweight Architectural Analytics](#), 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 blue planet studio from istock

### 24.5.1: 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 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.

### **24.5.2: Choose the right tool for the job**

Cassie Kozyrkov breaks down the techniques for analyzing data into three snazzy **groups**<sup>11</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 highlighting 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?" kinds of questions. Or "What is that service in our public cloud costing as \$1M per year?"
2. **Statistics:** Consider this your trusty Swiss Army knife for decision-making when dealing with incomplete information and uncertainty. For example, "How likely am I to get struck by lightning?" or "What is the probability of downtime for our IT services (famous three, four, five-nines of uptime)?"
3. **Machine Learning (ML)/AI:** This is your 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 will our IT costs be next year based on detailed traffic estimates?"

When you master these techniques, data transforms into your superpower, helping you ask sharper questions and deliver spot-on answers.

### **24.5.3: Prefer complete information to partial information**

No matter 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.

---

<sup>11</sup><https://kozyrkov.medium.com/what-on-earth-is-data-science-eb1237d8cb37>

Statistics is used when you don't have all the facts and must manage uncertainty. They can help you balance the likelihood of a wrong decision against your data budget, considering your risk preferences.

#### **24.5.4: 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 critical 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. Diving into data without asking the right questions is like wandering into a labyrinth with a blindfold on.

## 24.6: To Probe Further

- [Introduction to Decision Intelligence<sup>12</sup>](https://www.linkedin.com/pulse/introduction-decision-intelligence-cassie-kozyrkov/), by Cassie Kozyrkov, 2020
- [Decision Intelligence: The steering wheel for your life<sup>13</sup>](https://www.linkedin.com/learning/decision-intelligence/), by Cassie Kozyrkov, 2024
- [Making decisions right, even if they're not always the right decision<sup>14</sup>](https://architectelevator.com/strategy/always-be-right/), by Gregor Hohpe, 2021
- [Are IT's biggest decisions its worst?<sup>15</sup>](https://architectelevator.com/transformation/it-decisions/), by Gregor Hohpe, 2019
- [Your most important architecture decisions might be the ones you didn't know you made<sup>16</sup>](https://architectelevator.com/architecture/important-decisions/), by Gregor Hohpe, 2020

---

<sup>12</sup><https://www.linkedin.com/pulse/introduction-decision-intelligence-cassie-kozyrkov/>

<sup>13</sup><https://www.linkedin.com/learning/decision-intelligence/>

<sup>14</sup><https://architectelevator.com/strategy/always-be-right/>

<sup>15</sup><https://architectelevator.com/transformation/it-decisions/>

<sup>16</sup><https://architectelevator.com/architecture/important-decisions/>

## 24.7: 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 organizational complexity 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? 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?*
- *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?*

# 25: Economic Modeling With ROI and Financial Options: Learning From the Finance Field



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. However, answering it concisely and convincingly to a non-technical audience may be difficult.
- 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.

Decision-making in the corporate world is frequently an **economic risk exercise**. Financial and risk modeling is like the crystal ball of the corporate world. It helps organizations make intelligent decisions, like choosing between investing in a new project or finally fixing the office coffee machine. These models forecast financial performance and assess economic scenarios, ensuring companies aren't just throwing darts in the dark.

Economic and risk modeling is a game-changer in resource allocation. By predicting future trends and disruptions, these models help organizations **use their resources wisely**, for instance, to ensure they don't run out of coffee (a critical issue for any IT business). It's all about efficient resource management and avoiding those 'Oops, we should have seen that coming' moments.

These models majorly upgrade strategic planning, helping companies **anticipate challenges and opportunities** instead of reacting like a cat to a cucumber. Identifying risks before they bite means organizations can implement preventative measures, keeping things running smoothly.

Economic and risk modeling is the secret weapon for staying ahead of the game, achieving long-term goals, and ensuring the office party budget doesn't get blown on a single extravagant cake.

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.

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. Still, it may take much work to answer this seemingly harmless question concisely and convincingly to a non-technical audience without sounding like a techie version of Shakespeare.

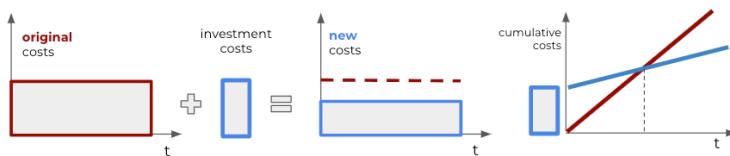
Good architecture requires some investment. This investment is time and effort spent implementing an architecture pattern, reducing technical debt, or refactoring code to align with our architecture. Consequently, we need to explain the expected value of this investment. It's all about showing that a little investment now will save a lot of headaches—and money—later.

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

## 25.1: 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).



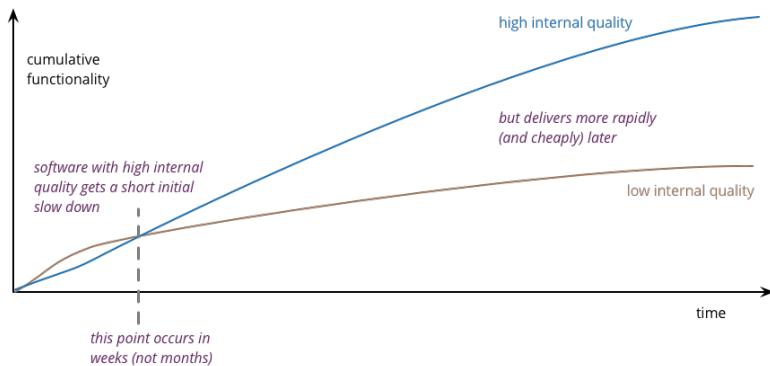
**Figure 1:** An illustration of the ROI metaphor. Investment leads to lower costs or higher value. It takes some time to reach a break-even point when an additional value has compensated for the investment. After the break-even point, we earn more than without the investment.

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

---

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



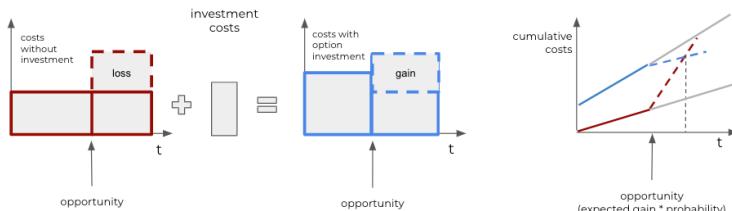
**Figure 2:** Software with high internal quality gets a short initial slowdown but delivers more rapidly and cheaply later (source [martinfowler.com/articles/is-quality-worth-cost.html](http://martinfowler.com/articles/is-quality-worth-cost.html)).

An ROI metaphor is easy to understand by a non-technical audience. Still, it has limitations when describing 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.

## 25.2: 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 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 exists for buying and selling financial options. Fischer Black and Myron Scholes computed 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 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

<sup>2</sup>[https://en.wikipedia.org/wiki/Black%E2%80%93Scholes\\_model](https://en.wikipedia.org/wiki/Black%E2%80%93Scholes_model)

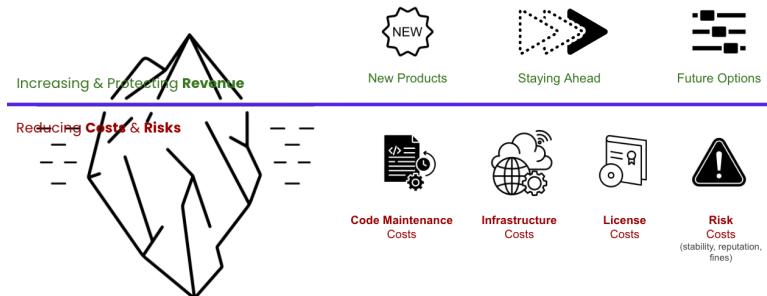
separates concerns allows changes to be localized so that updating one aspect of a system does not require expensive changing of the whole system. Such isolation will enable you to change a decision late in the project or even after go-live, at a nominal cost. For example, if authentication is a well-isolated concern, you must 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.

## 25.3: Communication Frameworks

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.

### 25.3.1: Top Line: Increasing and Protecting Revenue

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.

### **25.3.2: Bottom Line: Reducing Costs and Risks**

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. 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. Figure 5 illustrates what may happen if you do not invest. As 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, expensive cycle continues.

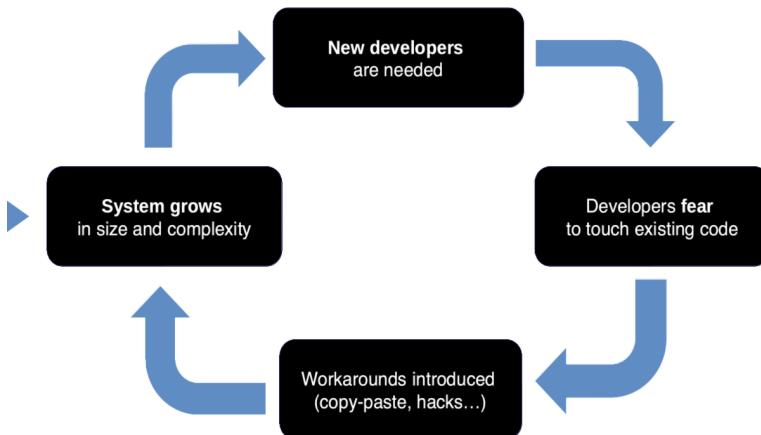


Figure 5: A downward spiral of poorly maintainable code.

**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.

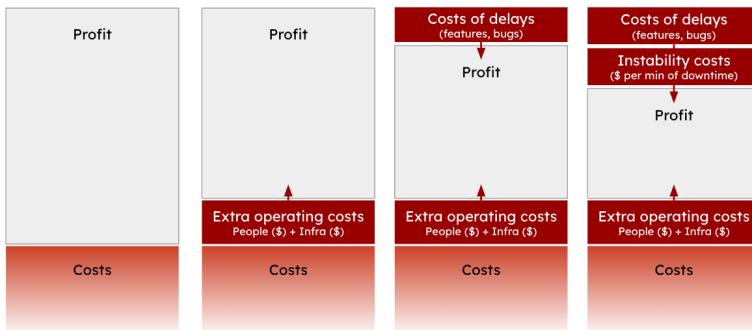
**Investments in reducing risk costs.** When your system is down, your business is disrupted, and you lose revenue. According to diverse studies<sup>3</sup>, the average cost of downtime ranges from \$2,300 to \$9,000 per minute. You must invest in keeping your system reliable and secure to avoid losing 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

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

and Sterman “Nobody Ever Gets Credit for Fixing Problems that Never Happened<sup>4</sup>”.

### 25.3.3: Example: The Tech Debt Reduction ROI Framework

In addition to the general framework discussion in this section, I frequently used the following framework to visually communicate how increasing technical debt progressively reduces profit through added operating, delay, and instability costs, emphasizing the importance of managing technical debt to preserve profitability.



**Figure 6:** A framework for discussing ROI of tech debt reduction from a profitability standpoint.

Figure 6 illustrates a framework for understanding the ROI (Return on Investment) of reducing technical debt, showing the impact on profit and costs across three progressive scenarios:

- 1. Extra Operating Costs:** Technical debt increases operating costs due to inefficiencies, reducing the profit.
- 2. Costs of Delays:** “Costs of delays” represent costs related to feature delays or bug fixing. These costs illustrate a further erosion of profit due to delays caused by technical debt, which impacts the ability to deliver new features or fix issues promptly.
- 3. Instability Costs:** Instability costs“ (e.g., cost per minute of downtime) add to “Costs of delays” and “Extra operating costs.” This

<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)

scenario shows the compounded effect of technical debt, where instability and downtime further reduce profit, creating multiple layers of costs and significantly impacting profitability.

This framework shows the compounded effect of technical debt, where extra operating costs increase the bottom line, and instability and downtime further reduce profit (reduce top line), creating multiple layers of costs and significantly impacting profitability.

## 25.4: To Probe Further

- Architecture: Selling Options<sup>5</sup>, by Gregor Hohpe, 2016
- Is High Quality Software Worth the Cost?<sup>6</sup>, by Martin Fowler, 2019
- Don't get locked up into avoiding lock-in<sup>7</sup>, by Gregor Hohpe, 2019

---

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

<sup>6</sup><https://martinfowler.com/articles/is-quality-worth-cost.html>

<sup>7</sup><https://martinfowler.com/articles/oss-lockin.html>

## 25.5: 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?*

# 26: How Big Transformations Get Done: Learning From Mega-Projects



image by elxeneize from istock

**IN THIS SECTION, YOU WILL:** Learn that IT transformation projects face similar challenges as other mega-projects, often failing due to cognitive biases and poor planning, but applying key lessons from successful projects—such as risk mitigation, modular design, and stakeholder engagement—can significantly improve their outcomes.

**KEY POINTS:**

- IT transformation projects are large-scale, multi-year endeavors that often aim for significant organizational changes, such as creating unified IT platforms after mergers, and they frequently face challenges similar to other mega-projects.
- Common cognitive biases and decision-making fallacies like overconfidence, strategic misrepresentation, and the sunk cost fallacy often derail these projects, leading to cost overruns, delays, and underperformance.
- Flyvbjerg and Gardner's book "How Big Things Get Done" highlights lessons from successful and failed mega-projects across various industries, offering 11 practical heuristics for improving project leadership, including hiring the right experts, building modular systems, and focusing on risk mitigation.
- Key strategies for IT success include taking the outside view to learn from similar projects, fostering strong stakeholder relationships, and avoiding unnecessary complexity or scope creep by staying focused on core objectives.

Most **IT transformations** involve an immense, multi-year, multi-million-dollar project. Contrary to continuous improvements, IT transformation projects frequently aim to create significant organizational changes, such as creating a single IT platform after merging dozens of companies. Due to their size, IT transformation projects belong to the **category of mega projects**. They can benefit from lessons that other mega projects have learned.

For practical strategies on how to navigate these challenges, '**How Big Things Get Done**' by Bent Flyvbjerg and Dan Gardner is an invaluable resource. This comprehensive exploration of large-scale projects not only identifies why they often fall short of expectations but also provides actionable steps to improve their success rates. Drawing from extensive research, real-world case studies, and psychological insights, the book offers a practical roadmap for better planning and execution, equipping people with the tools they need to succeed.

Flyvbjerg and Gardner discuss how project planners and stakeholders are

often **overly optimistic** or intentionally misrepresent facts to win approval, leading to budget overruns and delays. Large projects like airports, bridges, and Olympic infrastructure are notorious for going over budget and time, just like many IT projects. Flyvbjerg and Gardner explore how human psychology affects decision-making and project outcomes. They underscore the importance of fighting cognitive biases and fostering a culture of honesty and transparency in project planning, making it clear that these values are not just desirable, but necessary for successful project outcomes.

The book is a treasure trove of examples from successful and failed projects across various industries, including technology, construction, and entertainment. Contrasting examples like the successful **Pixar movie-making process** with troubled infrastructure projects provides a comprehensive understanding of the factors that contribute to project success or failure, enlightening the reader and informing their future decisions.

## 26.1: The Iron Law of Mega-Projects

Flyvbjerg and Gardner define the iron law of mega-projects as follows: “**Over budget, over time, under benefits, again and again.**” These words resonate with the **brutal reality** of mega-projects. Put simply, the typical project is one in which costs are underestimated and benefits are overestimated. Picture a big project that costs more than it was supposed to and delivers less than expected. Flyvbjerg and Gardner analyze famous examples, such as the [Sydney Opera House](#) (ten years late and 1,357% over budget<sup>1</sup>) and the [Big Dig in Boston](#), (nine years late and 190% over budget<sup>2</sup>) explaining why these problems are so widespread.

Here's what the data shows from their study of 16,000 such projects:

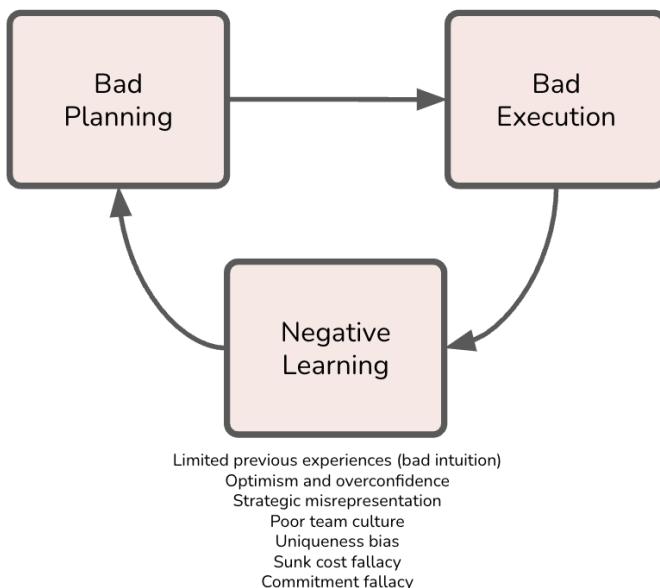
- A mere 0.50% hit budget, time, and benefit expectations (or better)
- 8.50% meet both budget and time targets (or better)
- 47.90% manage to stay on budget (or better)

That is, sadly, typical. On project after project, **rushed, superficial planning** is followed by a **quick start** that makes everybody happy because shovels are in the ground. But inevitably, the project crashes into **problems that were overlooked** or not seriously analyzed and dealt with in planning. People run around trying to fix things. More stuff breaks. There is more running around. Flyvbjerg and Gardner call this the “**break-fix cycle**”, like a “mammoth stuck in a tar pit.” Several cognitive biases and decision-making fallacies often sabotage these efforts, leading to cost overruns, delays, and underperformance. All these factors together frequently lead to a negative learning loop (Figure 1): “*The more you learn, the more difficult and costly it gets.*”

---

<sup>1</sup>[https://en.wikipedia.org/wiki/Sydney\\_Opera\\_House#Completion\\_and\\_cost](https://en.wikipedia.org/wiki/Sydney_Opera_House#Completion_and_cost)

<sup>2</sup>[https://en.wikipedia.org/wiki/Big\\_Dig](https://en.wikipedia.org/wiki/Big_Dig)



*Figure 1: The negative learning loop behind many mega-projects.*

For IT projects, Flyvbjerg and Gardner gave the following conservative estimates of base rates for cost risks based on their database of projects:

- Mean cost of overrun of IT projects is 73%
- 18% of IT projects belong to the long tail ( $\geq 50\%$  of overrun costs)
- Mean cost of overrun of IT projects in the long tail is 447%

This analysis means that even if you reserve an extra 100% of your planned budget for unforeseen problems, you still have a fair chance of ending up in the long tail, where costs could be arbitrarily high.

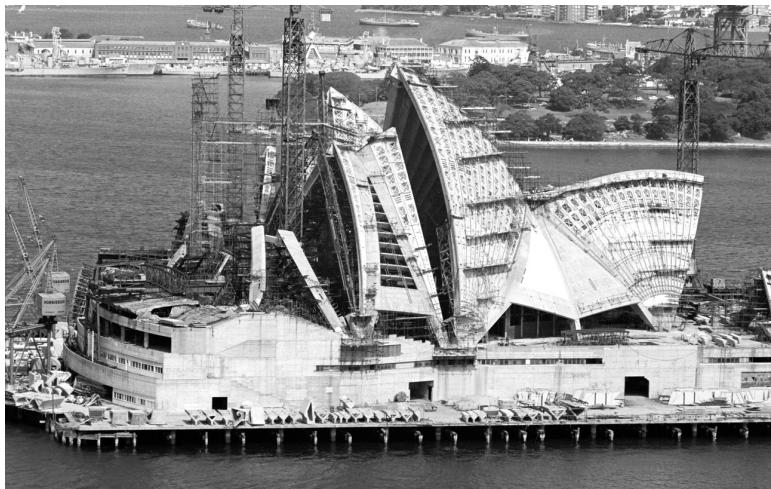


image by john carmen molla from istock

Let's explore some of typical mega-project pitfalls:

- Optimism and Overconfidence (Hope Is Not a Strategy)
- Strategic Misrepresentation
- Bad Team
- Uniqueness Bias
- Lack of Experience (Eternal Beginner Syndrome)
- Sunk Cost and Commitment Fallacies

### 26.1.1: Optimism and Overconfidence

Leaders and stakeholders frequently overestimate their ability to deliver complex transformations. Flyvbjerg and Gardner's key heuristic for managing optimism on projects is "*You want the flight attendant, not the pilot, to be an optimist.*"



image by skyneshher from istock

It's common for optimism and overconfidence to rear their heads in IT projects. Leaders and stakeholders may find themselves underestimating the complexity and risks involved. Here are some examples that you might find familiar:

- **Cloud Migration:** A company migrating from on-premise servers to a cloud-based infrastructure may **overestimate the skills** of its engineering team, assuming they can handle the complexity without significant retraining. This assumption can lead to unexpected integration challenges, especially when dealing with legacy systems that were not designed for cloud environments. As a result, timelines get extended, budgets are exceeded, and operational disruptions occur during the migration process.
- **Microservices Refactoring:** A software engineering team may decide to refactor their monolithic application into a microservices architecture, believing their existing codebase is stable enough to handle the transition seamlessly. They might **overlook the technical debt** accumulated in the monolithic system, such as tightly coupled components and poor documentation. This overconfidence in the system's stability can lead to unforeseen challenges, like managing inter-service communication, increased latency, or scaling issues, which delay the project and inflate costs.

- **AI Deployment:** In a project implementing AI or machine learning algorithms, a team might be overly optimistic about their ability to develop, train, and deploy the models effectively. They may ignore less predictable and the **time-consuming** processes of **data cleaning** and **model validation**, hoping the models will perform well immediately. This optimism can lead to delays when the models fail to deliver the expected results in production, forcing the team to go back and address foundational issues.

In all these cases, **overconfidence** can lead to a failure to properly **assess risks** and plan for the necessary resources, while **optimism** creates **unrealistic expectations** about timelines and project success. By underestimating early delays, leaders also fail to account for how delays compound over time, ultimately jeopardizing the entire project.

An interesting form of overconfidence is that **early delays are not seen as a big deal** by most project leaders. They figure they have time to catch up because the delays happen early. That sounds reasonable. But it's dead wrong. Early delays **cause chain reactions** throughout the delivery process. The later a delay comes, the less work there is and the less the risk and impact of a chain reaction. For instance, if a database migration is delayed by a few weeks due to integration issues, project managers may downplay the impact, assuming future phases can compensate for the lost time. However, these early delays often create a chain reaction, pushing back testing, deployment, and training phases, resulting in even greater delays and increased costs down the line. Or, as US President Franklin Roosevelt said, "*Lost ground can always be regained—lost time never.*"

### 26.1.2: Strategic Misrepresentation

In some cases, leaders knowingly present overly optimistic forecasts to get funding approval or to gain executive buy-in. This strategy avoids **rejection** from stakeholders who may be wary of the true timeline and costs.



image by catalin205 from istock

Here are some specific examples:

- **ERP System Overhaul:** A transformation project leader might present a rosy forecast for a large-scale Enterprise Resource Planning (ERP) system upgrade, promising that it will be completed within a year. Knowing that typical ERP implementations involve complex integrations with legacy systems, data migration, and extensive testing, the leader may hide these challenges to avoid scaring off executive stakeholders with a more realistic multi-year timeline. As the project progresses, the team quickly runs into unforeseen challenges, like data compatibility issues or workflow changes, which lead to delays, budget overruns, and frustration from the IT team and the executives who feel misled.
- **Software Refactoring:** A project manager might downplay the effort required to refactor a legacy system to make it more modular or scalable. They may minimize the technical debt associated with the old codebase—such as outdated libraries, hardcoded dependencies, or poor documentation—knowing that acknowledging these issues would require a larger budget and longer timeline. To secure immediate funding or developer resources, the manager assures leadership that the refactoring can be done quickly. However, as engineers dive into the project, they discover that cleaning up the

technical debt is far more complex than expected, leading to missed deadlines, increased costs, and team burnout.

- **Cloud Migration Misrepresentation:** A transformation project leader might promise that migrating a company's IT infrastructure to the cloud will result in immediate cost savings and improved efficiency within six months, downplaying the complexities of migrating legacy systems, rearchitecting applications for cloud-native services and addressing security concerns. Knowing that a more honest projection—perhaps 18 to 24 months—might reduce executive enthusiasm for the project, the CTO strategically presents an overly optimistic view. As the project unfolds, unexpected integration challenges arise, such as ensuring data consistency across cloud and on-prem environments, leading to delays and escalating costs that eventually sour stakeholders on the project.
- **Cybersecurity Overhaul:** An IT security leader might strategically underplay the scope of a cybersecurity overhaul, presenting the project as a relatively simple implementation of new tools, like firewalls and intrusion detection systems, while neglecting to mention the need for organization-wide security policy updates, employee training, and long-term monitoring. This underrepresentation might be used to secure quick buy-in from executives who are resistant to the idea of a larger-scale project. However, as the implementation begins, it becomes clear that these foundational changes are necessary, leading to scope creep, increased costs, and delays that frustrate both the project team and stakeholders.

In all of these cases, **strategic misrepresentation** might provide **short-term benefits** in the form of quick project approval and initial enthusiasm, but it sets the stage for **long-term failure**. Once the inevitable delays, cost overruns, and technical challenges surface, trust between leadership and the project team erodes, and the organization ends up paying a higher price—in both time and money—than if the project had been scoped and communicated accurately from the start.

### 26.1.3: Poor Team Culture

A strong, cohesive team is essential in mega-projects, where complexity and stakes are high. A **lousy team** not only **undermines performance** but also magnifies risks, leading to **cascading failures** that jeopardize the

entire project's success. In IT mega-projects a bad team can have severely negative impacts, amplifying the inherent complexities and risks.



image by charday penn from istock

When team members are disjointed, unmotivated, or lack trust in one another, several issues arise that can derail the project:

- **Poor Communication and Collaboration:** A lack of cohesion in the team leads to miscommunication, misunderstandings, and siloed efforts. This results in inefficient workflows, duplicated work, and critical information being missed or ignored, causing delays and costly rework.
- **Low Morale and Disengagement:** When team members do not feel empowered or connected to a shared goal, they become disengaged and less committed. This disengagement can lead to poor decision-making, suboptimal performance, and a lack of accountability, which in turn increases the likelihood of errors and project delays.
- **Inconsistent Quality and Execution:** A poorly structured team will struggle to maintain consistent work standards. Without a unified approach, varying levels of expertise and different working styles can create gaps in quality, leading to technical debt, integration issues, and unreliable systems that require additional time and resources to fix.

- **Increased Conflict and Blame Culture:** When team members are not aligned, conflicts over priorities, responsibilities, and blame for failures arise. This misalignment further damages morale slows decision-making and distracts from the core objectives, exacerbating delays and overruns.
- **Failure to Adapt to Challenges:** A dysfunctional team is less likely to adapt to inevitable project challenges, such as scope changes or unexpected technical issues. Without trust and a shared commitment to problem-solving, the team may be slow to respond to setbacks, resulting in missed deadlines, increased costs, and project failure.

Here are specific examples in IT projects where a bad team can severely impact execution:

- **Cloud Migration Project:** In a large-scale cloud migration project, where multiple teams (networking, security, application development, and operations) must work together, poor communication can lead to critical gaps. For example, if the networking team does not correctly communicate firewall rules and IP whitelisting requirements to the operations team, the migrated applications might fail to connect to necessary resources, leading to significant downtime and troubleshooting. If different teams use inconsistent approaches for migrating workloads (e.g., one team optimizes for performance while another focuses on cost), the result can be a fragmented cloud architecture, with some applications running smoothly and others facing performance bottlenecks or security vulnerabilities.
- **Microservices Architecture Refactor:** In a project to refactor a monolithic application into microservices, team members may resist change due to the complexity and additional workload. If developers feel unsupported or disconnected from the project's vision, they may not fully commit to learning new frameworks or technologies required for microservices. This situation can result in poorly designed services with limited scalability and high maintenance costs. Microservices require teams to adopt DevOps practices like continuous integration/continuous deployment (CI/CD). If the team is dysfunctional and does not fully embrace these practices, deployments may become manual and error-prone, leading to frequent downtime and slower release cycles.

- **Data Warehouse Implementation:** Conflicts can arise between data teams in a data warehouse project that consolidates data from multiple departments. If there is no alignment on data governance and standards, teams might blame each other for data inconsistencies or performance issues. For instance, the finance team may report issues with slow queries. In contrast, the engineering team insists that the database schema is not the problem, resulting in a standoff and no resolution. Different teams may load data into the warehouse with inconsistent validation checks. This inconsistency leads to corrupted or duplicate data, causing critical business reports to be inaccurate. Fixing these data quality issues after deployment would require additional resources, delays, and erode stakeholder trust in the system.
- **Agile Transformation for Software Development:** In an Agile transformation project, if teams do not trust or fully adopt Agile principles, such as frequent feedback loops and iterative development, they may revert to old habits of waterfall planning. This results in poor responsiveness to changing business needs missed sprint goals, and an inability to deliver incremental value. Agile ceremonies like sprint reviews or retrospectives without cross-functional collaboration become box-ticking exercises rather than meaningful ways to improve the process. Agile transformations require a shift in mindset. However, team members may become disengaged if the team feels that management is only paying lip service to the principles without making fundamental organizational changes (like reducing micromanagement). Disengaged teams miss opportunities to optimize workflows, introduce better tools, and improve velocity, ultimately stalling the transformation effort.
- **Custom CRM System Development:** In a project to build a custom CRM system for a large organization, poor collaboration between frontend and backend teams can result in misaligned system design. For example, the backend team might design inefficient or difficult APIs for the frontend team to use, causing delays in integration and forcing the frontend team to write additional code to make the system work as intended. If the development team lacks a unified testing strategy, some features may be well-tested while others contain bugs or performance issues that go unnoticed until production. This lack of strategy leads to frequent bug fixes post-launch, which frustrates users and increases long-term maintenance costs.

- **Cybersecurity Implementation in a Financial Institution:** In a cybersecurity overhaul, if the security team is not aligned with the development and operations teams, each group may prioritize its own tasks over shared security goals. For example, the development team might push out new features without proper security reviews, blaming the security team for delays, while the security team blames developers for introducing vulnerabilities. This conflict creates gaps in the system, exposing the organization to potential data breaches or compliance failures.

In these examples, a **lousy team dynamic**, characterized by poor communication, lack of engagement, inconsistency, and conflict, amplifies risks and delays in IT mega-projects. These issues affect technical outcomes and strain budgets, timelines, and stakeholder confidence.

#### 26.1.4: Uniqueness Bias

Uniqueness bias means people see their projects as unique, one-off ventures with **little or nothing to learn from earlier projects**. If you imagine that your project is so different from other projects that you have nothing to learn from them, you will **overlook risks** you would catch and mitigate if you switched to the outside view instead.



image by yuri\_arcuri from istock

IT transformations are often seen as unique by those executing them, leading to a reluctance to learn from other industries or similar projects. However, the reality is that lessons from other sectors, such as construction or film production, can be valuable. Here are some examples of how this bias plays out and its consequences:

- **Custom Enterprise Solution Development:** A company developing a custom enterprise resource planning (ERP) solution might believe its specific business processes are so unique that it cannot use existing frameworks or off-the-shelf solutions. As a result, the company might build the entire solution from scratch rather than adopting and customizing proven ERP systems. This “reinventing-the-wheel” approach leads to unnecessary complexity and development delays. Established ERP systems often have pre-built modules for common business processes (finance, HR, supply chain), and trying to recreate these from scratch introduces risks of inefficiencies, bugs, and a longer time to market. By failing to learn from existing systems, the company overlooks best practices in data management, integration, and compliance, increasing the risk of costly rework or system failure.
- **Cloud Migration Project:** During a cloud migration, IT leaders might believe that their company’s legacy systems and data architecture are so unique that they require a completely custom approach to the cloud strategy. This view leads them to ignore established cloud migration frameworks or case studies from other industries that have faced similar challenges. By not learning from similar cloud migrations, the team may overlook common risks, such as data migration errors, service downtime, or security vulnerabilities. Established frameworks, like those provided by cloud service providers (AWS, Azure), offer proven processes for phased migration, testing, and scalability. Still, a team with a uniqueness bias might dismiss these, leading to project delays and potential operational disruptions.
- **DevOps Implementation:** A software engineering team tasked with implementing a DevOps pipeline might believe their existing software development environment is so specialized that common DevOps tools and practices won’t work. They might create custom deployment and testing pipelines, ignoring widely adopted tools like Jenkins, Docker, or Kubernetes. Building custom solutions

increases the project's complexity and adds long-term maintenance costs. Commonly used DevOps tools are battle-tested, scalable, and come with extensive community support. By rejecting these tools, the team risks developing systems that are less reliable, more challenging to scale, and more costly to maintain, all of which could have been avoided by adopting industry-standard solutions.

- **Security Overhaul in a Financial Institution:** A financial institution undergoing a cybersecurity overhaul might view its security needs as so unique due to regulatory requirements that it avoids using best practices or tools that have been widely adopted in the industry. Instead, they may build custom security protocols or ignore lessons learned from other institutions' security breaches. This decision could lead to increased vulnerability to attacks or compliance failures. Lessons from other industries, such as incident response plans, network segmentation, or automated threat detection, would provide a robust framework for addressing risks. By failing to learn from others, the institution might introduce avoidable weaknesses into its security system, making it more prone to breaches or compliance penalties.
- **Overlooking Project Management Practices:** A large-scale software engineering project might view itself as fundamentally different from other industries, such as construction or manufacturing, dismissing project management practices commonly used in those sectors. Teams might underestimate the value of phased delivery, risk management frameworks, or using a project's "critical path" to prioritize tasks. Ignoring lessons from other fields, the team fails to implement effective risk mitigation strategies, leading to unexpected scope creep, budget overruns, and delays. In contrast, large infrastructure projects often use planning techniques that account for risks and dependencies, allowing for better control of timelines and costs. Adopting similar methodologies could help IT teams manage complexity and reduce uncertainty.

In each of these cases, **uniqueness bias** prevents teams from benefiting from proven strategies and lessons learned in similar or even unrelated fields. By adopting the **outside view** and learning from both similar IT projects and other industries, teams can avoid common pitfalls, reduce risk, and improve project success rates.

### 26.1.5: Lack of Experience (Eternal Beginner Syndrome)

When an organization embarks on its first large-scale IT transformation, it often lacks experience in managing complex systems and technologies. This **lack of institutional memory** leads to **repeated mistakes** that more experienced teams could have avoided. For instance, in an IT architecture overhaul, the team might struggle to implement a scalable infrastructure due to poor planning or inadequate tools despite established best practices in the industry. In software engineering, teams might make the same coding mistakes, like inadequate version control or failure to manage dependencies properly, because they lack a disciplined software development lifecycle (SDLC).



image by skynesh from istock

Here are some examples illustrating this issue:

- **IT Architecture Overhaul:** A company undergoing its first IT architecture overhaul to implement a microservices-based architecture may lack experience in designing scalable systems. The team might underestimate the complexity of decoupling tightly integrated legacy systems, failing to plan for key elements such as inter-service communication, data consistency, and fault tolerance.

Without adequate planning and knowledge of best practices, the company may end up with a fragile architecture that suffers from frequent outages or performance bottlenecks. Experienced teams would have anticipated the need for load balancing, service discovery, and failure recovery mechanisms, but the lack of experience leads to significant downtime and costly rework to stabilize the system.

- **Data Migration to the Cloud:** A company with no prior experience in cloud migration may attempt to move its entire on-premise data infrastructure to a cloud provider without fully understanding the implications of bandwidth constraints, data security requirements, or cloud-native application design. They might fail to implement proper backup strategies or to account for the potential latency issues with large datasets. The lack of experience leads to failed migrations, with critical data being lost or corrupted in the process. Additionally, unoptimized applications and data structures cause excessive cloud costs and slow performance. These issues could have been mitigated with a more experienced approach, such as phased migration, cost management tools, or cloud optimization practices.
- **DevOps and CI/CD Implementation:** An organization attempting to adopt DevOps practices for the first time might struggle to establish continuous integration (CI) and continuous deployment (CD) pipelines due to a lack of familiarity with the necessary tools and automation. They may fail to properly set up test automation, monitoring, or infrastructure as code (IaC). Without a clear understanding of DevOps principles, the team ends up with manual deployment processes prone to human error, resulting in slower releases, frequent production issues, and high maintenance costs. Experienced teams would have prioritized automating the build, test, and deployment processes, ensuring more reliable and faster iterations.
- **Security and Compliance in IT Systems:** A company handling sensitive customer data for the first time might lack the necessary experience to implement proper security and compliance protocols, such as encryption, role-based access control, or regular security audits. The lack of institutional knowledge on data protection leads to significant vulnerabilities, exposing the organization to data breaches or non-compliance with regulations like GDPR or HIPAA.

Experienced teams would have implemented security best practices from the outset, ensuring that data is protected and compliance requirements are met.

- **Technical Debt Accumulation:** Inexperienced teams might focus on short-term goals and quick feature releases without considering long-term scalability or maintainability. As a result, they accumulate significant technical debt with complex, poorly documented code that is difficult to maintain or extend. This technical debt slows down future development and increases the likelihood of bugs and system crashes, as the underlying infrastructure is not built to handle new demands. More experienced teams would have balanced feature development with addressing technical debt, ensuring that the codebase remains maintainable and scalable.

In these examples, **Lack of Experience and Eternal Beginner Syndrome** leads to costly mistakes and delays. Organizations embarking on their first IT transformation often overlook best practices and proven methodologies, resulting in inefficiencies, poor performance, and technical failures. Organizations can avoid these repeated mistakes by learning from more experienced teams or consulting external experts and achieving more successful outcomes.

### 26.1.6: Sunk Cost and Commitment Fallacy

The **Sunk Cost Fallacy** and **Commitment Fallacy** are closely related cognitive biases that lead decision-makers to continue investing in a project, often well past the point of viability, to justify previous investments or out of reluctance to admit failure. Together, they create a cycle of sustained investment in projects that may be outdated, inefficient, or otherwise unworkable.



image by georgeclerk from istock

The **Sunk Cost Fallacy** occurs when organizations persist in allocating resources—time, money, or effort—based on what has already been invested rather than the project’s current and future benefits. Meanwhile, the **Commitment Fallacy** or **Commitment Bias** arises when decision-makers feel compelled to push forward with a failing initiative, fearing that abandoning it would mean admitting defeat or wasting prior commitments. This push often results in quick “lock-in” decisions that overlook alternatives and elevate project risk.

Here are examples of how these biases manifest:

- **Custom Data Warehouse vs. Cloud Solutions:** A company may continue investing in a costly, on-premises data warehouse, even when cloud-based solutions offer better performance and scalability. The justification lies in the large initial investments in infrastructure, which overshadow the inefficiency and higher maintenance costs compared to cloud alternatives. Instead of shifting to a cloud solution, the company experiences mounting operational inefficiencies.
- **Legacy Codebase in Obsolete Languages:** A software team may continue enhancing an outdated codebase due to the significant resources already spent on development and training. Despite

modern frameworks offering greater ease of maintenance and scalability, the team resists change, ultimately leading to a bloated, hard-to-maintain system that falls behind in performance.

- **Over-customized ERP Systems:** Companies that heavily customize ERP systems to fit their needs face issues when updates or integrations become difficult. Despite the challenges and escalating costs, they continue investing in these customizations due to prior investments, ultimately impeding the company's flexibility and competitiveness compared to more adaptable ERP options.
- **In-house Software vs. Off-the-Shelf Solutions:** Companies often invest in building custom solutions when off-the-shelf options are available. Even as maintenance becomes costly and new features are slow to implement, they avoid switching because of prior development costs, risking inefficiency and missed opportunities to streamline operations.

These fallacies lead to several detrimental impacts:

- **Inefficient Resource Use:** Continued investment in outdated or failing projects wastes valuable resources.
- **Increased Project Risk:** Projects suffer from technical debt, rigidity, and escalating costs that increase long-term risks.
- **Missed Opportunities:** By resisting modern, flexible solutions, companies miss out on enhanced scalability, cost savings, and competitive advantage.

To avoid these biases, organizations should adopt a data-driven approach that includes regular, objective reassessments of projects based on current value and future potential, not past investments. This approach can enable timely pivots to alternative solutions, reducing long-term costs and fostering innovation. Recognizing when to re-evaluate, abandon, or adapt projects is essential for sustainable growth and competitiveness.

## 26.2: Heuristics for Successful Mega-Projects

Cognitive biases and fallacies represent a significant risk in IT transformations. Overcoming them requires **rigorous active planning**, the **willingness to learn** from other industries and past mistakes, and a focus on making data-driven decisions. Successful IT transformations often involve **transparency, adaptability**, and a **willingness to pivot** when necessary rather than sticking to flawed assumptions, unrealistic optimism, or sunk investments.



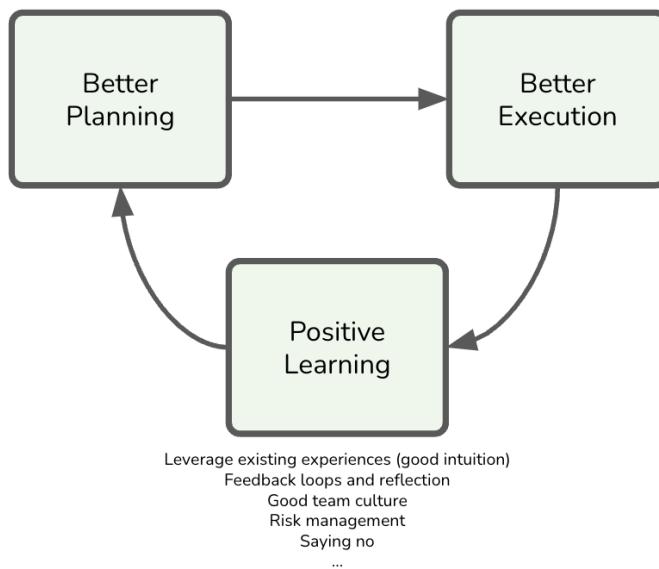
image by skynesh from istock

Flyvbjerg and Gardner do not only analyze projects but have identified 11 heuristics for better project leadership:

- Hire a Masterbuilder
- Get Your Team Right
- Ask Why
- Build With Lego
- Think Slow, Act Fast
- Take The Outside View
- Make Friends and Keep Them Friendly

- Watch Your Downside
- Know That Your Biggest Risk Is You
- Say No and Walk Away
- Build Climate Mitigation Into Your Project

These techniques can set a project on the right path, creating a positive learning loop that pushes projects forward and improves over time.



*Figure 2: The positive learning loop.*

### 26.2.1: Hire Experienced Masterbuilders

In any complex IT project, whether designing a new system, migrating to the cloud, or building a software platform, having **exemplary leadership** can make or break the effort. Flyvbjerg and Gardner claim that hiring a **master builder**—with deep expertise and hands-on experience—should be your top priority. The term originates from the skilled masons who built Europe’s medieval cathedrals, individuals with both the vision and the practical knowledge to bring massive, intricate projects to life. In today’s

world, it means finding someone with a **track record of successfully leading** similar IT initiatives.

It is important to understand what Flyvbjerg and Gardner mean by a master builder. I interpret their heuristics for a master builder as someone who **isn't necessarily controlling every detail upfront** but rather someone who has relevant experience and can **lead change effectively**, ensuring the project evolves successfully in real-time while balancing both strategic and operational priorities. A master builder is not necessarily a top-down architect who rigidly controls every aspect of a project. Instead, a master builder can be any key figure who knows how to drive change in an unpredictable, complex environment. This role is less about dictating every decision and more about **enabling the project to adapt and succeed** amidst constant flux, instilling confidence in the team and stakeholders.

In IT projects, a master builder could be a **Chief Technology Officer (CTO)**, **Chief Product Officer (CTO)**, a **Chief Architect**, a **Senior Product Manager** (or a team of them)—any senior leader with strong, relevant experience, the deep expertise and strategic vision required to steer the team through unforeseen challenges. They don't merely impose a rigid vision; they understand the technical and business intricacies, foresee potential risks, and know how to pivot when things don't go as planned. Essentially, they act as adaptive leaders who empower their teams and navigate uncertainty, keeping the project aligned with its goals while responding to emerging challenges.



image by freshsplash from istock

For instance, if you're overseeing a major system integration, your master builder might be a **seasoned CTO, chief architect, or principal engineer** who's navigated the complexities of combining legacy systems with modern cloud services. They'll know the potential pitfalls—like data migration challenges or API compatibility issues—and how to avoid them. Their domain experience allows them to **anticipate risks**, adapt to changes, and keep the project on track.

But, as you may have experienced, master builders **aren't always available or affordable**, especially for smaller projects or organizations with limited budgets. If that's the case, you'll need to think creatively. You might look for someone with solid expertise but lacks the full experience of a master builder. In this case, you'll want to **build a strong support network** around them, such as hiring subject matter experts to supplement their knowledge or bringing in external consultants for critical phases of the project.

Another option is to foster internal talent by **investing in mentorship and training**. With the proper guidance, you may have **rising stars** on your team who can step into a master builder role. Over time, you build the expertise you need internally by allowing them to lead smaller projects or shadow more experienced colleagues.

In any case, whether you hire an established expert or grow your own, the

key is to ensure that someone with deep domain knowledge is at the helm of your project. The master builder has the practical wisdom to ensure your project's success, guiding it through the technical and managerial challenges that inevitably arise.

### 26.2.2: Get Your Team Right

This heuristic is the one that every successful project leader will tell you is **non-negotiable**: *Get your team right*. As Ed Catmull, co-founder of Pixar, wisely said, “*Give a good idea to a mediocre team, and they will screw it up. Give a mediocre idea to a great team, and they will either fix it or come up with something better. If you get the team right, chances are they will get the ideas right.*” This statement holds especially true in IT, where complex systems, evolving technologies, and tight deadlines make teamwork essential to success.



image by simonkr from istock

But who should assemble the team? Ideally, this should be the responsibility of a *master builder*. Picking the right team is arguably the master builder's most important job. Whether you're building a new cloud infrastructure, implementing a security overhaul, or leading a software development project, the right team can overcome technical obstacles, innovate on the fly, and adapt to changing requirements.

In IT projects, the master builder's role is far from solitary. It's their deep expertise and **leadership that sets the tone**, but it's the **team that delivers the project**. For example, when designing a distributed architecture, the master builder would bring together the right mix of cloud architects, developers, DevOps engineers, and security experts, ensuring that someone fills each role with the technical skills and collaborative mindset to succeed. A well-assembled team has the technical chops and the ability to work together smoothly, troubleshoot issues, and stay focused on the project's overarching goals.

Let's take an example in software engineering: You may have a brilliant principal engineer (the master builder) who can architect an elegant solution. However, if the team around them lacks the necessary talent in areas like front-end development, API integration, or QA testing, the project is likely to fall short. The master builder's real genius lies in their technical knowledge and ability to pick, mentor, and manage a team that works together cohesively toward the project's success.

What if a master builder isn't available? In that case, the project manager needs to **focus on team composition**. They must evaluate not just each member's technical skills but also how well those individuals collaborate. In IT, a team that communicates well and adapts to new challenges will far outperform one technically brilliant but siloed or resistant to change.

Remember, teams, not individuals, deliver projects. So, to amend the earlier advice: *When possible, hire a master builder—and the master builder's team*. Their combined expertise, working in sync, will ultimately drive your project to success. Even when faced with a less-than-perfect plan, a great team can adapt, innovate, and deliver far beyond expectations.

### 26.2.3: Ask Why

Asking why you're doing your project is one of the most powerful ways to **stay focused on what really matters**—your ultimate purpose and the desired result. IT projects often get bogged down in technical details, changing requirements, or unforeseen obstacles, constantly returning to the core “*Why?*” keeps you **grounded in the bigger picture**. This overarching goal should be placed in the metaphorical “box on the right” of your project plan—a constant reminder of the result you're striving for.



image by courtneyk from istock

For example, if you’re leading a project to migrate systems to the cloud, the ultimate purpose is improving scalability and reducing operational costs. As the project progresses and technical challenges arise—such as data migration delays, compatibility issues, or security concerns—it’s easy to get lost in the details. But a strong leader continuously asks, “*Why are we doing this?*” If your actions, resources, or decisions don’t contribute directly to achieving the project’s purpose, they need to be reconsidered.

For instance, when implementing a microservices architecture, getting caught up in the technical nuances of service design, API management, or deployment pipelines is easy. However, if the ultimate goal is improving system flexibility and development speed, you must evaluate every technical decision against that goal. This strategic evaluation ensures that every technical decision is goal-oriented and contributes directly to the desired outcome.

Asking “*Why?*” It can also help you **prioritize features** and **manage scope creep**. As your project evolves, stakeholders may request additional features or changes. Before agreeing to any modification, ask, “*Why is this necessary?*” And “*Does this change help achieve the primary goal?*” If the answer is no, it may be best to push back or deprioritize the request in favor of what drives the project toward success. This approach gives you a sense of control and decisiveness in managing the project’s scope.

By continuously asking “*Why?*” throughout the life of your project, you ensure that every decision and every action is contributing to the ultimate result. This helps you stay focused on delivering value and prevents you from getting sidetracked by details that don’t serve the bigger picture.

### 26.2.4: Build With Lego

**Big is best built from small.** Just like how a towering wedding cake is essentially a stack of smaller cakes, many large-scale projects in IT and engineering are constructed from simple, repeatable building blocks. This modular approach allows you to **scale systems efficiently**, getting **better, faster, and cheaper as you grow**. Think of each small unit as a **Lego brick**—a fundamental building block that, when replicated, creates something far more powerful.



image by albertpego from istock

In IT, this principle is at the core of many successful architectures. For instance, consider **server farms**: the server is the Lego brick. A data center isn’t a massive, monolithic structure; it’s a collection of individual servers stacked and connected in ways that allow them to scale up or down as needed. Similarly, in cloud computing, **virtual machines** or **containers** act as Lego bricks, providing modular, repeatable computation units that can be easily replicated or distributed across a global network.

The same idea applies to **microservices architecture** in software development. Instead of building one massive, monolithic application, you create a series of small, independent services that communicate with each other. Each microservice is a Lego brick—focused, self-contained, and scalable. If one service needs to handle more traffic, you can scale it independently without affecting the rest of the system. This modular approach makes your application more flexible, maintainable, and easier to scale.

This idea isn't limited to software and server architecture. Take **infrastructure as code (IaC)**—where you define your IT infrastructure using code templates, automating the setup and deployment of systems. Each piece of infrastructure—a network configuration, a virtual machine, or a storage unit—can be considered a Lego brick. With IaC, you can deploy, replicate, and scale your infrastructure quickly and consistently by reusing and stacking these blocks.

Modularity also applies to **software development processes**. Agile methodologies break down considerable development efforts into small, manageable sprints, where teams build, test, and release incremental pieces of functionality. Each sprint is like adding another Lego brick to your larger product, allowing continual improvement and flexibility.

This approach has been successfully applied beyond IT—whether in solar farms (where each **solar panel** is a building block), in shipping (where **containers** are standardized units), or even in **education** (where modular learning systems allow customization and scalability). Its applicability is only limited by imagination.

Building your systems, software, or infrastructure with Lego-like components creates the flexibility and scalability needed for long-term growth. Each piece is small, manageable, and optimized independently, but they form something much more significant together. The beauty of this approach is that it's not only scalable but adaptable to changes and innovations, allowing you to improve incrementally as your needs evolve.

### 26.2.5: Think Slow, Act Fast

Taking the time to **plan thoroughly** is crucial because the stakes during execution are much higher. As the saying goes, “measure twice, cut once.” What’s the worst that can happen during planning? You may lose some code snippets or adjust the project roadmap. But what’s the

worst that can occur during delivery? Your data migration corrupts critical information, causing system-wide failures. Your untested API causes a security breach—your cloud infrastructure crashes during peak demand, costing millions in downtime. Almost any nightmare scenario can happen—and has happened—during delivery. That's why it's essential to limit your risk exposure.

You do this by **thinking slowly** during the active planning phase. In this stage, the **cost of reworking** a system design or rethinking your technology choices is **relatively low**. You can test ideas, simulate failures, and anticipate potential issues with little consequence. Thinking through every detail—designing the architecture for a new platform, setting up cloud infrastructure, or defining security protocols—pays off massively when it's time for action.



image by nanostockk from istock

Again, it is essential to understand what Flyvbjerg and Gardner mean by thinking slowly. Planning and thinking slowly is neither a theoretical exercise nor a traditional waterfall approach where you decide everything up front without room for adjustment. Instead, it's a practical, iterative, and dynamic process that involves experimenting, simulating, exploring, and doing everything necessary to clarify your roadmap, reduce risks, and minimize surprises. In IT architecture and software engineering, this means testing different approaches, running simulations to expose potential failures, and experimenting with configurations to see what works best. The goal is to refine your strategy and eliminate uncertainties before you hit the high-stakes phase of full-scale delivery.

Unlike rigid waterfall planning, this approach encourages continuous feedback, real-time learning, and adaptability. For example, in planning a system migration or platform launch, you're not just brainstorming once and sticking to a static plan; you're actively testing workloads, simulating failure scenarios, and exploring alternative solutions as new information becomes available. This experimentation gives you a clear understanding of what to expect, where the risks are, and how to navigate them before you commit to the real-world rollout.

This approach ensures you're not just “thinking” slow—you're adjusting and learning to avoid the costly surprises that can derail a project during full-scale delivery. It's an approach that balances thoughtful planning with the flexibility to adapt.

Flyvbjerg and Gardner use Pixar's planning process as an example of an exemplary process for mega-projects. It is well-known for being highly iterative, collaborative, and flexible, allowing the team to explore ideas in depth before settling on a final narrative or visual direction. Here's a breakdown of how it works:

1. **Brainstorming and Concept Development:** Every project begins with a core idea, often a story or theme pitched by a director or team member. From here, Pixar holds brainstorming sessions to flesh out the concept, characters, and plot possibilities.
2. **Storyboarding and Story Reels:** Pixar artists create rough storyboards to visualize scenes. The artist then assembles storyboards into “story reels,” like animated storyboards that give a sense of pacing and flow. This early visualization helps the team assess how scenes work together and identify potential issues with the story structure.
3. **Brain Trust Reviews:** One of Pixar's unique processes is the “Brain Trust.” This trust is a group of experienced directors, writers, and creatives who meet periodically to review the story's progress. They offer feedback, critique, and insights to help the director improve the story. The Brain Trust isn't there to dictate; instead, it provides unfiltered feedback, allowing the director to decide how to move forward.
4. **Iteration and Refinement:** Based on feedback, the team revisits storyboards and story reels, constantly refining and reworking scenes. Pixar embraces this iterative approach, repeatedly rethink-

ing and reshaping the story until it resonates emotionally and makes narrative sense.

5. **Early Animation Tests and Visual Development:** Pixar experiments with rough animation and visual styles early on to see how characters will look and move. They may conduct tests with lighting, colors, and textures, exploring different visual approaches that align with the story's tone.
6. **Technical Development and Simulations:** Pixar's stories often require innovative animation techniques, so the technical team is involved early to develop tools, simulate effects, and experiment with new technologies. This step is akin to running simulations in software development to ensure their tools and techniques will work before total production.
7. **Pre-Production and Production:** Once the story is solidified, Pixar moves into pre-production, where it finalizes designs, builds models, and rigs characters. By this point, Pixar is confident in the story and its direction, having refined it through constant testing, feedback, and adjustment. The project then moves into full production, creating the final animations.
8. **Continued Feedback and Tweaks:** Even during production, Pixar maintains flexibility. They continue to screen the movie to the Brain Trust and even test audiences, using this feedback to make necessary adjustments. They're open to late-stage tweaks, ensuring the final product is polished and compelling.

This iterative planning and production model has been integral to Pixar's success. It allows them to refine their ideas meticulously and collaboratively while being open to creative shifts throughout the process. It balances deep, thoughtful exploration with a flexible approach that adapts to new insights and discoveries along the way.

For example, suppose you're planning a migration from on-premise servers to the cloud. In that case, meticulous planning can help you identify which workloads should move first, how to maintain system continuity, and how to test for issues before going live. Spending extra time to plan and test may seem slow, but it's a **fraction of the cost and hassle** compared to dealing with unanticipated failures during delivery. It can also reduce the chances of needing a complete rollback or emergency patch, which can be expensive and damaging to your reputation.

You act fast once the planning is complete and the **team understands the roadmap**. Delivery is where the real risks lie. Every decision made during delivery carries higher stakes—downtime, security breaches, data loss, or even failure to meet critical deadlines. The key is minimizing the window in which things can go wrong by executing precisely and quickly. In a software deployment, this means having automated testing and CI/CD pipelines in place so that when you act, you can quickly detect and fix issues before they spiral out of control.

Consider launching a new app or major system upgrade. During the planning phase, you've mapped out user journeys, tested scalability, and ensured the system can handle peak loads. Once you go live, acting fast means monitoring performance in real time, quickly resolving bottlenecks, and rolling out patches swiftly if issues arise. You want the period of uncertainty—when the system is most vulnerable—to be as short as possible.

Good planning boosts the odds of a quick, effective delivery. It allows you to act decisively, execute smoothly, and close the window on risk as fast as possible. Thinking slow and acting fast is how you control complex projects, limit the dangers, and ensure a successful outcome.

### 26.2.6: Take The Outside View

Your project may feel **unique and special**, but it is likely **part of a larger class of projects** unless you are working on something that has never been done before—like developing an AI that achieves accurate general intelligence or creating a quantum computing network. Getting caught up in the specifics of your application or system can be easy. However, thinking of your project as “**one of those**” (for example, one of many cloud migrations, microservices refactor, or platform integrations) will allow you to gather data and learn from the collective experience of similar initiatives.



image by eoneren from istock

**Reference-class forecasting** is a powerful tool here. For instance, if you’re planning a cloud migration, you can look at data from other companies or projects that have done the same thing. How long did it take them? What common issues did they face, like underestimating data transfer times or managing service disruptions? By using this data, you’ll make more accurate predictions about your own project’s timeline and resource needs.

The same applies when adopting a new technology stack. Say your team is switching from a monolithic architecture to microservices. While this might feel like a radical shift, there’s a wealth of data from others who’ve made this transition. You can learn from the time it typically takes to decompose services, the common pitfalls like latency introduced by inter-service communication, and the operational challenges of managing a distributed system.

**Risk mitigation** also benefits from this outside view. For example, if you’re implementing an API-first strategy, it’s easy to focus on the specific requirements of your API consumers. But take the outside view and look at other API-first projects. You might discover risk patterns—such as API versioning issues, backward compatibility, or security concerns—that many projects face. Recognizing these risks early on, because they are common across projects of this nature, will help you address them proactively.

By shifting your focus from your project's uniqueness to the class of projects it belongs to, you'll have a **more accurate understanding of what to expect**. The broader perspective will allow you to make better-informed decisions about timelines, costs, and risks. Ultimately, embracing this paradoxical approach leads to better project outcomes.

### 26.2.7: Make Friends and Keep Them Friendly

A leader of a multibillion-dollar public sector project once told the authors he spent more than half his time **acting like a diplomat**, fostering the understanding and support of stakeholders who could significantly influence his project. Why? Because **it's risk management**. Stakeholder relationships can be as critical to project success as technical proficiency.



image by caiaimage /sam edwards from istock

Consider an IT project to build a company-wide data lake. From the CTO to the heads of various business units and the security team, each stakeholder has a vested interest in the project. The CTO is focused on innovation and delivery, business unit leaders are concerned about how it will impact their workflows, and the security team worries about data governance and compliance. If one of these groups becomes disengaged or adversarial, it could stall the project at a critical juncture. By taking the time to understand and address each stakeholder's concerns early,

you **create allies** who are more likely to help resolve issues when they inevitably arise.

Take another example: integrating a legacy system with a new ERP platform. The legacy system might be critical to a department that feels threatened by change. Building relationships with the department head, listening to their concerns, and involving them in key decision-making processes will ensure their cooperation when the integration hits a snag, whether related to data compatibility or performance issues.

When a project hits a roadblock—a budget overrun, a technical issue, or a timeline delay—the strength of your relationships with stakeholders can determine whether you’ll get the support you need to solve the problem or face resistance. And **when something goes wrong, it’s too late to start** building those relationships. If the data lake suffers performance issues or the ERP integration causes workflow disruptions, you’ll want those stakeholders in your corner, ready to collaborate instead of criticize.

The lesson is clear: **build your bridges before you need them**. Invest time cultivating goodwill and understanding with those who can impact your project. Their support could be the key to turning a potential project failure into a success.

### 26.2.8: Watch Your Downside

It’s often said that opportunity is as important as risk. However, this is a misconception. The truth is, **risk can significantly disrupt your project, and no potential upside can compensate** for catastrophic failure. This is particularly crucial in the face of fat-tailed risks—rare but severe issues that can have a disproportionate impact. Instead of relying on forecasting risk probabilities, your focus should be on risk mitigation. Failure to manage these risks could lead to severe consequences for your project.



image by glenn hewitt from istock

For example, consider a project to migrate a company's entire infrastructure to the cloud. On paper, the opportunities are massive—scalability, cost savings, and improved performance. However, the risks, such as data loss during migration, misconfigurations that lead to security vulnerabilities, or unexpected downtime, could spell disaster. It doesn't matter how great the benefits are if a critical outage compromises your customer's trust or results in costly compliance violations. A wise project leader in this situation would focus first on ensuring robust data backup strategies, rigorous testing, and failover systems before getting excited about the potential upside of cloud performance.

Another example is implementing a microservices architecture. While microservices offer the promise of flexibility and scalability, it's crucial to remember the downside risks. These include service failure due to mismanagement of dependencies, network latency issues, or even cascading failures across services. The key here is not to get carried away by the agility and speed of deploying new features, but to first ensure that the system is resilient. This means building robust monitoring, automating redundancy, and preparing for the worst-case scenarios, such as service outages or database failures.

Flyvbjerg and Gardner use the example of a rider in the grueling three-week Tour de France bicycle race who explained that participating is

not about winning but not losing, day after day for twenty-one days. Only after that can you consider winning. Successful IT project leaders think the same way; they focus on not losing—every day—by mitigating risks such as data breaches, performance bottlenecks, or regulatory non-compliance. Once you manage the risks, you can start thinking about optimizing for success.

In short, successful project leaders prioritize downside protection. They understand that the most significant opportunity doesn't matter if unchecked risks derail the entire effort. They focus on reducing risk and “not losing” each day while keeping an eye on the prize—the ultimate goal they are trying to achieve.

### **26.2.9: Know That Your Biggest Risk Is You**

It's easy to think that projects fail because of external factors—scope changes, unforeseen technical issues, new regulations, or shifting management priorities. While these challenges certainly arise, they are not the root cause of most project failures. In reality, many projects fail because we fall **prey to our biases and blind spots**. Jim Lasko's Great Chicago Fire Festival didn't fail because he couldn't predict the specific malfunction of the ignition system—it failed because he didn't take the *outside view*, which would have shown him how failure typically occurs in live events. He focused on his project's unique details and ignored the broader lessons from similar events.



image by martin barraud from istock

In IT, this **inside view** bias can manifest in many ways. A project manager might assume that a new system will integrate smoothly with existing infrastructure because they've seen their team solve integration challenges before, ignoring that the vast majority of similar projects encounter delays or failures during this phase. Or a software engineer might believe they can implement a cutting-edge technology stack because they're excited about its potential, disregarding the typical risks associated with introducing unproven tools.

**Behavioral biases** like overconfidence, optimism bias, and scope neglect can derail even the most well-intentioned project leaders. For instance, it's common to underestimate the time and resources needed to complete a complex software build, leading to missed deadlines and blown budgets. You might believe that your team is different or that your project is unique, but this thinking ignores the wealth of data from countless failed projects that followed the same pattern.

In software development, for example, how often have ambitious timelines slipped because someone believed the team could handle it? Or how often has scope creep been justified by the belief that "just one more feature" won't cause significant delays? These are classic inside-view mistakes. To avoid them, take the outside view—consider your project as part of a larger class of similar efforts. How do these projects typically

fail? What risks do they usually face? Use historical data, reference-class forecasting, and learn from the failures of others. This learning process can empower you to make better decisions and improve your project outcomes.

A practical example: if you're leading a project to implement a new enterprise resource planning (ERP) system, resist the temptation to believe that your team will avoid the common pitfalls because they're talented or because this project feels different. Look at how ERP projects typically fail—data migration issues, resistance to change, misalignment with business processes—and plan accordingly. Acknowledge that these risks apply to your project, too, and focus on mitigating them from the start.

By acknowledging that your biggest risk is you—your biases, assumptions, and overconfidence—you can make better decisions. You can mitigate risks by looking beyond the specifics of your project and learning from the broader class of similar projects. This shift in perspective not only helps you avoid common pitfalls but also opens up a world of potential success. It can make the difference between a successful project and a failed one.

### **26.2.10: Say No and Walk Away**

Staying focused is critical for getting projects done, especially in IT projects, where complexity can spiral quickly. The ability to say no is essential for maintaining that focus. At the outset of any project, you must assess whether you have the people, the budget, and the contingencies needed to succeed. If the answer is no, you must have the discipline to walk away, or at the very least, reconsider or delay the project until the conditions are right.



image by deepak sethi from istock

Saying no early on protects you from a variety of pitfalls. For example, if a proposed system integration lacks an adequate budget or experienced engineers, pushing ahead will likely result in delays, technical debt, or outright failure. Declining or rescoping the project is better than proceeding with insufficient resources. Similarly, during the project, continuously ask: *Does this action directly contribute to achieving the goal?* If it doesn't, skip it. This discussion means saying no to **monuments**—features or systems that look impressive but add no real value—or to **untested technologies** that might introduce unnecessary risk without a clear payoff.

One common temptation in software development is **adopting the latest and greatest technology**, even when it's **not mature or well-suited** to the project. This temptation can lead to unnecessary complexity and time spent troubleshooting rather than building. If the technology doesn't directly support your core objectives, say no and stick with proven solutions. Likewise, you should say no to **scope creep**—the gradual expansion of a project beyond its original goals. It's easy to get sidetracked by requests for new features or integrations, but every additional element increases complexity, introduces new risks, and stretches your resources.

Saying no can be especially difficult in organizations with a **bias for action**, where the pressure is on to deliver and move fast constantly. But

success often comes from what you don't do. Steve Jobs once said, "I'm as proud of the things we haven't done as the things we have done." Apple's ability to stay focused on just a few products allowed them to perfect those products and achieve massive success. In the same way, IT projects succeed when teams focus on doing fewer things but doing them well.

In other cases, saying no isn't just about scope or technology—it's about risk. For instance, if a project exposes your company to potential legal issues (such as data privacy concerns or regulatory non-compliance), saying no early on is critical. Even if the project seems valuable, the risk of lawsuits or regulatory fines may far outweigh any potential benefits. Recognize when it's time to walk away.

Ultimately, **saying no is about discipline**. It keeps the team focused on the primary goal, preserves resources, and ensures that the project stays on track. While saying no can feel difficult at the moment, especially when everyone is pushing to do more, it's often the key to long-term success. By cutting out distractions, avoiding unnecessary risks, and focusing on what truly matters, your project and your organization will be better positioned to succeed.

### **26.2.11: Build Climate Mitigation Into Your Project**

No task is more urgent today than mitigating the climate crisis. **The time to act is now**, not only for the common good but for your organization, yourself, and your family. In the realm of IT projects, climate action should be a core consideration for any project you undertake.



image by francesco scatena from istock

For IT and software engineers, this might mean focusing on **energy efficiency** in data centers, applications, and infrastructure. For example, consider not just performance and cost but also the environmental impact when designing a cloud architecture. Choose cloud providers that prioritize renewable energy in their data centers. Many leading cloud platforms, such as AWS, Google Cloud, and Microsoft Azure, have committed to sustainability goals and offer carbon-neutral options. By selecting these, you reduce the carbon footprint of your project while maintaining its technical excellence.

Another practical step is **electrifying everything** and ensuring that your IT infrastructure's electricity comes from renewable sources. This step could involve shifting to more energy-efficient hardware and systems that consume less power. In software, you can contribute by optimizing code and reducing unnecessary computation, which cuts energy usage. If your organization has its data centers, you can advocate for a shift to renewable energy sources, such as solar or wind, to power them.

A concrete example might involve designing a green software project from the ground up. This involves adopting principles like sustainable coding, where algorithms are optimized for lower energy consumption, and using tools to monitor your application's energy impact. Additionally, consider how AI and machine learning models are deployed—these technologies

are notoriously resource-intensive. Implement energy-efficient practices for training and running models, such as using specialized hardware designed for energy efficiency or scheduling resource-heavy processes when renewable energy supply is highest.

We already have the technology and knowledge to implement these strategies—what's left is scaling them up across thousands of projects, large and small. This is where you, as an IT leader or engineer, play a vital and empowering role. Every system you design, every platform you build, and every line of code you write can contribute to a more sustainable future. Whether it's through reducing waste, minimizing power consumption, or supporting renewable energy, the opportunities for climate mitigation in IT are vast.

Climate mitigation should no longer be seen as an optional “nice to have” in project planning. It's a fundamental part of responsible engineering and project management. Integrating sustainability into your IT initiatives helps accelerate the global transition to a low-carbon future. Following the principles laid out in this book, you can ensure your projects succeed on a technical level and contribute to the greater good.

## 26.3: To Probe Further

- How Big Things Get Done<sup>3</sup>, by Bent Flyvbjerg and Dan Gardner, 2023
- The Mythical Man-Month<sup>4</sup>, by Fred Brooks, 1975
- Sketching User Experiences: Getting the Design Right and the Right Design<sup>5</sup>, by Bill Buxton, 2007

---

<sup>3</sup><https://www.amazon.ca/How-Big-Things-Get-Done/dp/077109843X>

<sup>4</sup>[https://en.wikipedia.org/wiki/The\\_Mythical\\_Man-Month](https://en.wikipedia.org/wiki/The_Mythical_Man-Month)

<sup>5</sup><https://www.amazon.com/Sketching-User-Experiences-Getting-Design/dp/0123740371>

## 26.4: Questions to Consider

- *What strategies can be employed to avoid strategic misrepresentation and ensure that initial project forecasts are honest and achievable?*
- *In what ways can IT leaders apply the lessons learned from other industries (e.g., construction, entertainment) to improve the success rates of large IT projects?*
- *How can organizations identify and address “sunk cost” and “commitment fallacies” in their IT transformation projects?*
- *What are the qualities of a “master builder” in IT transformations, and how can organizations ensure they have the right leadership in place?*
- *How can IT teams stay focused on the core “why” of the transformation and avoid being sidetracked by unnecessary features or scope creep?*
- *How can organizations apply the “build with Lego approach” in IT projects to create scalable, modular systems?*
- *How can taking an outside view, or reference-class forecasting, help IT leaders better estimate timelines, costs, and risks in their projects?*
- *What strategies can IT leaders use to foster strong, collaborative relationships with stakeholders to support project success?*
- *What are the risks of failing to “watch your downside” in IT transformations, and how can organizations better prioritize risk mitigation?*
- *What role do cognitive biases, particularly overconfidence, play in IT transformation failures, and how can teams safeguard against them?*
- *How can climate mitigation be integrated into IT transformation projects, and why is this essential for the organization and society?*

# **Part V: Summary**

## 27: Summary



image by ruhey from istock

**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 practice 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.

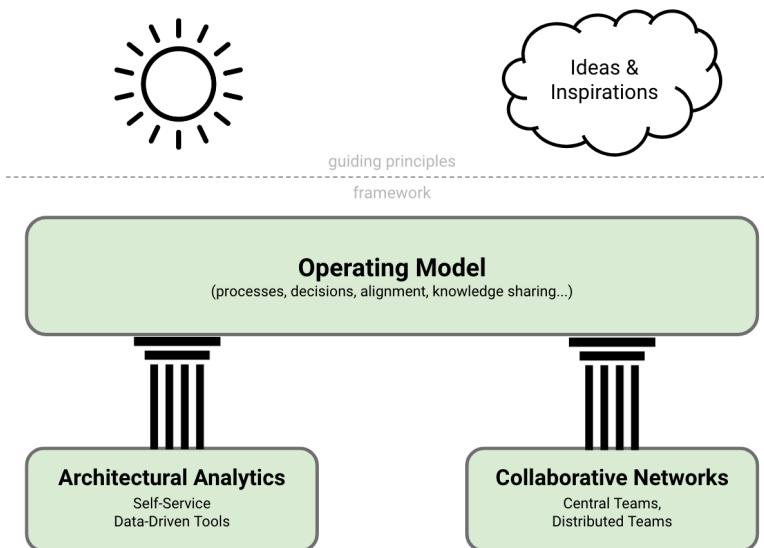
This book shared my approach to running an IT architecture practice in larger organizations based on my experience at AVIV Group, eBay Classifieds, and Adevinta. I called this approach “Grounded Architecture”—architecture with strong foundations and deep roots. Prioritizing people interactions and data over processes and tools, Grounded Architecture aims to connect an 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 (Figure 1):

- **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.

The Grounded Architecture framework consists of three elements:

- **Lightweight Architectural Analytics**,
- **Collaborative Networks**,
- **Operating Model**.



*Figure 1: Grounded Architecture Overview.*

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

*Collaborative Networks* is another essential element of Grounded Architecture. A strong network of people doing architecture across the organization is crucial to ensure that an architecture practice has any tangible impact.

Lastly, the sections *Operating Model: General Principles*, *Cooperation-Based Operating Model*, and *IT Governance: Nudge, Taxation, Mandates* define a collection of processes and agreements that lets architects do their thing, leveraging data and people connections to create a powerful, 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 several parts:

- **On Being an Architect:**

- Building Skills
- Making Impact
- Leadership
- Thinking Like an Architect: Architects as Superglue
- Thinking Like an Architect: Balancing Curiosity, Doubt, Vision, and Skepticism
- Architects' Career Paths

- **On Human Complexity:**

- The Culture Map: Architects' Culture Compass
- The Human Side of Decision-Making
- Effortless Architecture

- **Expanding the Architect's Toolkit: Learning From Other Fields:**

- Architecture in Product-Led Organizations: Learning From Customer-Centric Fields
- Decision Intelligence in IT Architecture: Learning From Data, Social, and Managerial Fields
- Economic Modeling With ROI and Financial Options: Learning From the Finance Field
- How Big Transformations Get Done: Learning From Mega-Projects

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 an architecture practices, I always advise not to forget these **two pieces of advice from Gregor Hohpe<sup>1</sup>:**

---

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

- “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 an architecture practice. You can’t just plug some architects into the current mess and expect it to solve all your problems.”

## 28: To Probe Further: Online Appendix Overview



image by pexels from pixabay

**IN THIS SECTION, YOU WILL:** Get an overview of the resources in the appendix.

As we bring this book to a close, I would like to equip you with a helpful appendix. These resources 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.

Inspiration and Further Learning:

- **Favorite Quotes**<sup>1</sup>: a selection of my favorite quotes about architecture.
- **Bookshelf**<sup>2</sup>: an overview of the background work to probe further, linking several external resources inspiring my work.

Tools for Developing Soft Skills:

- **Resources for Managing, Growing, and Hiring Architects**<sup>3</sup>: Pointers to resources for managing, growing and hiring architects. When growing and hiring architects, it is crucial to continually raise the bar, ensuring that the team is composed of highly skilled and diverse individuals who can contribute unique perspectives and expertise.
- **Effective Communication**<sup>4</sup>: This section synthesizes various resources specifically designed to enhance your ability to communicate effectively, provide constructive feedback, and manage challenging conversations with confidence and clarity. These resources are not just theoretical, but they provide practical strategies and techniques that you can apply in your daily interactions.
- **Resources for Working With Toxic Colleagues**<sup>5</sup>: This section summarizes how challenging personalities like the Kiss-Up/Kick-Down, Credit Stealer, and Bulldozer in high-pressure environments can undermine team dynamics. However, techniques such as documenting contributions and fostering transparency can help manage their negative impact.

---

<sup>1</sup><https://grounded-architecture.io/quotes>

<sup>2</sup><https://grounded-architecture.io/bookshelf>

<sup>3</sup><https://grounded-architecture.io/career-resources>

<sup>4</sup><https://grounded-architecture.io/communication>

<sup>5</sup><https://grounded-architecture.io/toxic-colleagues>

- **Resources for Dealing With Scapegoating at Work**<sup>6</sup>: This section summarizes how professionals in high-pressure environments, such as IT and software engineering, can avoid unfair blame for systemic issues through strategies like clear communication, documentation, and fostering shared accountability.

#### Pragmatic Knowledge Resources:

- **EIC/ISO 25010 Standard**<sup>7</sup> 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**<sup>8</sup> offer a mix of crucial distributed system and messaging system topics combined with modern public cloud engineering themes.
- **Characteristics of High Performing Organizations**<sup>9</sup> 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.

#### Software Tools:

- **Software Tools**<sup>10</sup> an overview of several tools I’ve built and use in daily architectural work.
- **Software Tools: Examples and Screenshots**<sup>11</sup> screenshots of concrete tools I built as a part of Lightweight Architectural Analytics websites.
- **Building Lightweight Architectural Analytics**<sup>12</sup> a few practical tips on building lean architecture dashboards and documents (e.g., to create Lightweight Architectural Analytics) using simple, widely available tools.

---

<sup>6</sup><https://grounded-architecture.io/scapegoating>

<sup>7</sup><https://grounded-architecture.io/iso25010>

<sup>8</sup><https://grounded-architecture.io/cloud-design-patterns>

<sup>9</sup><https://grounded-architecture.io/high-performing-organizations>

<sup>10</sup><https://grounded-architecture.io/tools>

<sup>11</sup><https://grounded-architecture.io/screenshots>

<sup>12</sup><https://grounded-architecture.io/data-website>