



Grounded Architecture

Redefining IT Architecture Practice in the Digital Enterprise

Željko Obrenović

Grounded Architecture

Redefining IT Architecture Practice in the Digital Enterprise

Željko Obrenović

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

This version was published on 2024-06-27



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

1: Introduction	1
1.1: What Will You Learn?	5
1.2: Key Influences	9
1.3: Why This Book?	10
1.4: A Part of the Bigger Picture: A Trilogy in Four Parts	11
1.5: A Bit of Personal History	13
1.6: The Structure of the Book	14
1.7: Stay Connected	15
1.8: Acknowledgments	16
2: Context: Fast-Moving Global Organizations	17
2.1: Global Scale	19
2.2: Multi-Dimensional Diversity	21
2.3: Nonlinear Growth Dynamics	24
2.4: Synergy and Transformation Pressures	26
2.5: Decentralization and Loose Coupling	28
2.6: Questions to Consider	30
3: Evolution of Architecture: Embracing Adaptability, Scalability, and Data-Driven Decisions	31
3.1: Goals	34
3.2: Questions to Consider	37

CONTENTS

Part I: Grounded Architecture	38
4: Grounded Architecture: Introduction	39
5: Data Foundation	42
5.1: Examples of Data Foundation Tools	45
5.2: Requirements For A Data Foundation	55
5.3: Building Data Foundation	57
5.4: Using Architecture Data Foundation	59
5.5: Appendix: Examples of Insights From Source Code Analyses	61
5.6: Questions to Consider	65
6: People Foundation	66
6.1: Background: Central vs. Federated Architecture Function	69
6.2: The Hybrid Model	71
6.3: Building People Foundation	76
6.4: Questions to Consider	78
7: Architecture Activities Platform	79
7.1: Examples of Architecture Activities	82
7.2: Guiding Principles for Architectural Excellence: Policies, Autonomy, and Engagement	84
7.3: Setting Boundaries	92
7.4: Questions to Consider	95
8: Transforming Organizations with Grounded Architecture	96
8.1: Executing at Scale	98
8.2: Adaptivity	101
8.3: Improving the Quality of Decision-Making with Data	104
8.4: Maximizing Organizational Alignment	107
8.5: Maximizing Organizational Learning	110
8.6: Questions to Consider	114

CONTENTS

Part II: Being Architect	115
9: Being Architect: Introduction	116
10: Architects as Superglue	118
10.1: Supergluing in Action: Reducing Tension among Business Functions, Product, Technology, Organi- zation	121
10.2: Superglue Abilities	126
10.3: Questions to Consider	130
11: Skills	131
11.1: Hard Skills	133
11.2: Soft Skills	135
11.3: Product Development Skills	137
11.4: Business Skills	138
11.5: Decision-Making Skills	139
11.6: Questions to Consider	140
12: Impact	141
12.1: Pillars of Impact	143
12.2: Questions to Consider	147
13: Leadership	148
13.1: David Marquet’s Work: The Leader-Leader Model .	150
13.2: Netflix’s Valued Behaviors: Leadership Behaviors .	153
13.3: Questions to Consider	158
14: Architects’ Career Paths: Raising the Bar	159
14.1: Typical Architect’s Career Paths	161
14.2: Hiring Architects	163
14.3: Questions to Consider	166

CONTENTS

Part III: Doing Architecture: Inspirations	167
15: Doing Architecture: Introduction	168
16: The Culture Map: Architects' Culture Compass	171
16.1: Communicating	174
16.2: Evaluating	176
16.3: Persuading	178
16.4: Leading	180
16.5: Deciding	182
16.6: Trusting	184
16.7: Disagreeing	186
16.8: Scheduling	188
16.9: Rules	190
16.10: Questions to Consider	191
17: Managing Organizational Complexity: Six Simple Rules	192
17.1: Background: Limitations of Hard and Soft Management Approaches	195
17.2: Six Simple Rules Overview	197
17.3: Rule 1: Understand What Your People Do	199
17.4: Rule 2: Reinforce Integrators	201
17.5: Rule 3: Increase the Total Quantity of Power	203
17.6: Rule 4: Increase Reciprocity	205
17.7: Rule 5: Extend the Shadow of the Future	207
17.8: Rule 6: Reward Those Who Cooperate	209
17.9: Questions to Consider	211
18: Effortless Architecture	212
18.1: Effortless State	216
18.2: Effortless Action	229
18.3: Effortless Results	245
18.4: The Road to Effortless Achievement	262
18.5: Questions to Consider	263

CONTENTS

19: Effective Communication	264
19.1: So What!	267
19.2: Radical Candor	276
19.3: Never Split the Difference	282
19.4: Questions to Consider	289
20: Understanding Product Development	290
20.1: The Build Trap	294
20.2: The Discipline of Market Leader	301
20.3: Product Operations	305
20.4: Questions to Consider	310
21: Architecture Governance: Nudge, Taxation, Mandates	311
21.1: Nudging	314
21.2: Taxation (Economic Incentives)	317
21.3: Mandates and Bans	319
21.4: Questions to Consider	321
22: Economic Modeling: ROI and Financial Options	322
22.1: The Return-on-Investment Metaphor	325
22.2: The Financial Options Metaphor	327
22.3: A Communication Framework	329
22.4: Questions to Consider	333
23: Decision Intelligence in IT Architecture	334
23.1: Basics of Decision-Making	337
23.2: Preparing for Making Decisions	341
23.3: Decision-Making Complexity	346
23.4: Decision-Making With Data and Tools	351
23.5: Questions to Consider	354
24: The Human Side of Decision-Making	355
24.1: Biases and Limitations	357
24.2: Indecisiveness	370
24.3: Intuition	372
24.4: Group Decision-Making Dynamics	376

CONTENTS

24.5: Questions to Consider	382
Part IV: Wrapping Up	383
25: Summary	384
26: Cheat Sheet	388
26.1: Introductions	389
26.2: Grounded Architecture: Introduction	391
26.3: Being Architect: Introduction	393
26.4: Doing Architecture: Introduction	395
Part V: To Probe Further	399
27: Bookshelf	400
27.1: Introduction	401
27.2: Career Development	404
27.3: Hard Skills	405
27.4: Soft Skills	411
27.5: Organization and Processes	416
27.6: Business, Product, Strategy	418
28: Tools	420
29: Favorite Quotes	423
29.1: People, People, People	424
29.2: Changing Role of Architecture	425
29.3: Complexity	426
29.4: Other	427
Part VI: Appendix	428
30: Appendix Overview	429

31: ISO 25010 Standard	431
31.1: Overview	432
31.2: Maintainability	433
31.3: Security	435
31.4: Performance Efficiency	437
31.5: Reliability	439
32: Cloud Design Patterns	441
32.1: Overview	442
32.2: Performance and Scalability	443
32.3: Resiliency	444
32.4: Messaging	445
32.5: Management and Monitoring	446
32.6: Security	447
32.7: Other Patterns	448
33: High Performing Technology Organizations	449
33.1: Overview	450
33.2: Four Key Metrics	451
33.3: Practices	452

1: Introduction



image by fda54 from pixabay

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

KEY POINTS:

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

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 just 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 name “Grounded Architecture” is a hat-tip to the [Grounded Theory](#)¹ methodology. Grounded Theory is about developing theories straight 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 some dusty old book of abstract principles.

¹https://en.wikipedia.org/wiki/Grounded_theory

This book breaks down Grounded Architecture into two main parts:

- **Structure:** 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 shows the structure of the Grounded Architecture consisting of three elements:

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

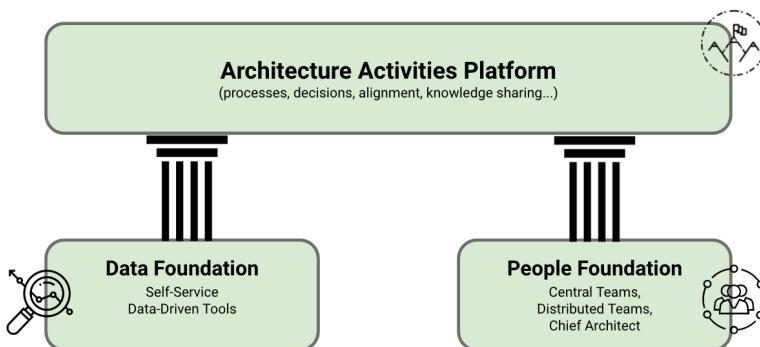


Figure 1: The structure of Grounded Architecture.

The *Data Foundation* keeps your architecture decisions based on up-to-date and complete information about your organization's tech landscape.

The *People Foundation* represents a strong network of people working on architecture across your organization, ensuring your efforts make a difference.

Lastly, the *Architecture Activities Platform* defines a collection of processes and agreements that lets architects do their thing,

leveraging data and people 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. These resources are grouped into two parts and are sure to make your journey both insightful and entertaining:

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

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

1.1: What Will You Learn?

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

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

1.1.1: General Philosophy

I approach designing an architecture practice in complex organizations as an art of cooking. Modern IT architecture practice is 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. That's why I've organized this book into modules you can mix and match in your own kitchen, just like a chef selects ingredients to create a mouthwatering dish.

In this culinary adventure, I'll share 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 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 basic structures and practices might be the same across different business scenarios, the most critical elements—like fresh, local ingredients in cooking—must be sourced locally. The staff's skills, experiences, and insights are the secret ingredients that customize and perfect the architecture to meet specific business needs and goals. They're what make the architecture truly bespoke and effective.

So, tie on your apron, sharpen your knives, and prepare to cook up an IT architecture practice that's as unique and satisfying as a gourmet meal.

1.1.2: Format

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

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.

1.1.3: Content

This book is **not technical**. We will not discuss the details of public cloud design patterns, security, reliability, how to optimize computer loads, or select the proper data storage. As a modern architect, you will need these skills, 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.

1.1.4: Is This A Proven Method?

Like with many similar books, you may be disappointed if you are looking for a scientifically proven “method” of running a modern

architecture practice. This book is **personal and opinionated**, building on my daily experiences as an architect and head of an architecture practice. While subjective, this book can provide valuable insights for IT architects, their managers, and people working with architects. I have successfully **applied my approach in three different companies**, which gives it some generality and repeatability.

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

1.1.5: Who Should Read This Book?

When writing this book, I had a broad audience in 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: Key Influences

The Grounded Architecture approach also builds on many ideas that others have successfully used. Gregor Hohpe's **Architecture Elevator**² 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**³ and Will Larson's book **Staff Engineer: Leadership beyond the management track**⁴ are helpful guides in defining the responsibilities of modern architects. Overall, the Staff-plus engineering roles provide excellent examples of the development of architects.

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

²<https://architectelevator.com/>

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

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

1.3: Why This Book?

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

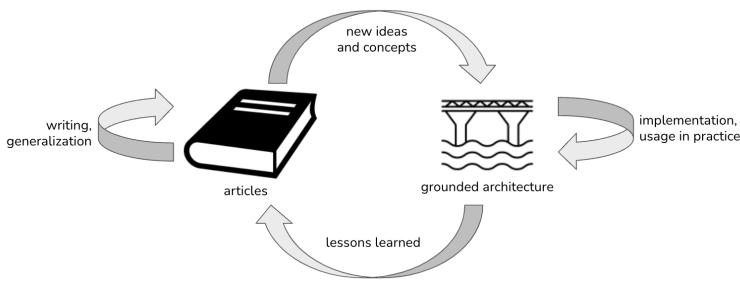


Figure 2: Writing a book helped me organize ideas, obtain new insights, improve principles and tools, and share the lessons learned.

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.

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

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

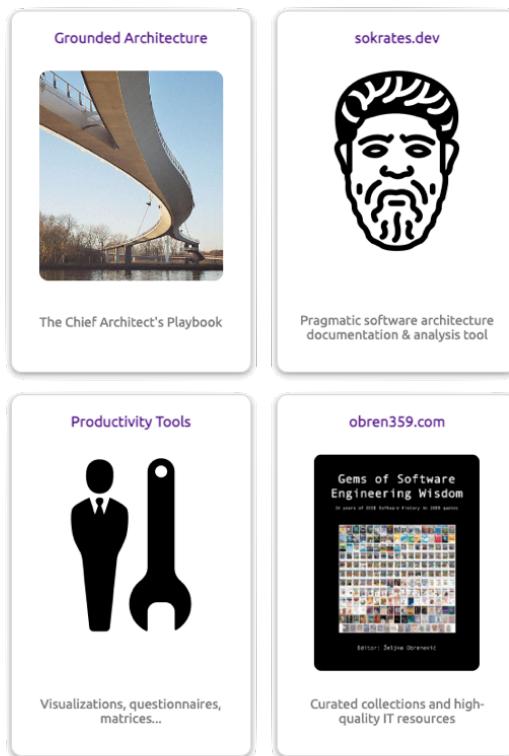


Figure 3: Grounded Architecture is a part of the collection of open-source tools and resources I have built in the past ten years to help me in architectural work.

⁵<https://obren.io/>

The other resources include:

- **Sokrates**⁶: an open-source polyglot code analysis tool. Sokrates provides a pragmatic, inexpensive way to extract rich data from source code repositories. Sokrates can help you understand your code by making the size, complexity, and coupling of software, people interactions, and team topologies visible.
- **Productivity Tools**⁷: A collection of more than 100 online tools I built to help me in my daily work as an architect.
- **359° Overview of Tech Trends**⁸ 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.io⁹.

⁶<https://sokrates.dev>

⁷<https://obren.io/tools>

⁸<https://www.obren359.com/>

⁹<https://obren.io/>

1.5: A Bit of Personal History

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

Another vital part of my experience that shaped this book was my earlier experience as a **consultant and analyst** at the **Software Improvement Group**¹⁰. 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**¹¹, which enables efficient and pragmatic extraction of data about technology and organization from source code. This work has directly influenced my view of the architecture **Data Foundation**.

My experience as a **CTO** of **Incision**¹², 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)**¹³ and **Eindhoven Technical University (TU/e)**¹⁴ 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**¹⁵, that I co-authored with Erik Stolterman, where we connected experiences of designing/architecting in three disciplines: classical design, UX design, and software engineering. This work has helped me better relate to and learn from non-technical fields.

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

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

¹¹sokrates.dev

¹²<https://incision.care>

¹³<https://www.cwi.nl/en/>

¹⁴<https://www.tue.nl/en/>

¹⁵<https://design-instability.com/>

1.6: The Structure of the Book

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

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

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

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

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

1.7: Stay Connected

You can find additional resources online at:

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

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

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

¹⁶<https://grounded-architecture.io/>

1.8: 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 Nesciobrug¹⁷](#). Credit: the botster, CC BY-SA 2.0, via Wikimedia Commons.



image by henk monster cc by 3 0 via wikimedia commons

¹⁷https://commons.wikimedia.org/wiki/File:Nesciobrug_4.jpg

2: Context: Fast-Moving Global Organizations



image by paul brennan from pixabay

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

KEY POINTS:

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

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

To truly grasp any idea or solution, you need to understand the context in which these ideas were born. I base my views on my time as a Chief Architect at AVIV Group and as a Principal Architect at eBay Classifieds and Adevinta. Let's dive into some key characteristics of the organizational context that shaped my Grounded Architecture approach:

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

2.1: Global Scale

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

2.2: Multi-Dimensional Diversity

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

- **Cultures:** Collaborated with a varied workforce and clientele, both local and remote.
- **Organization:** Engaged with units of different sizes, complexities, and organizational styles.
- **Product:** Managed diverse features catering to various markets and customer segments.
- **IT Architecture:** Balanced legacy systems with modern approaches.
- **Technology:** Utilized numerous programming languages and thousands of third-party libraries, frameworks, and services.



image by simon from pixabay

I worked with units differing in several aspects:

- **Unit Size:** Ranged from hundreds of employees to just a dozen.
- **Team Topologies:** Spanned from single-team setups to hierarchical team organizations.
- **Architectural Roles:** Varied from having dedicated local architecture teams and lead architects to smaller units where team members handled architectural duties alongside other responsibilities.

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 (AWS, GCP, Azure) and custom-built private data centers. Our systems employed a variety of application technologies, such as:

- **Database Technologies:** MySQL, PostgreSQL, MongoDB, Cassandra, AWS RDS, and more.
- **Backend Programming Languages:** Java, C#, Go, Scala, PHP, Node.js, Kotlin, among others.
- **Mobile App Programming Languages:** Swift, Objective-C, Java, Kotlin, Flutter/Dart, and more.
- **Frontend Programming Languages and Frameworks:** React, AngularJS, Vue, jQuery, and others.

Diversity offers several opportunities, including:

- **Increased Technology Innovation:** A diverse workforce can explore more technologies and tools creatively.
- **Better Implementation:** Access to a broader pool of resources allows for selecting the best tool for the job.

However, diversity also brings challenges:

- **Increased Complexity:** Higher system landscape complexity and cognitive load for teams mastering numerous topics simultaneously.
- **Reduced Flexibility:** Expertise is spread across many domains and technologies, limiting reorganization possibilities.
- **Higher Technical Debt:** Multiple technology stacks can lead to increased legacy components and outdated technologies.

In conclusion, while diversity is a rich source of new possibilities from an architectural perspective, it also necessitates managing complexity carefully.

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 technol-

ogy stacks. Furthermore, these nonlinear dynamics **necessitate a flexible architecture** to accommodate potential divestitures.

In summary, while nonlinear growth offers substantial benefits, it also presents challenges in managing increased complexity and maintaining architectural flexibility.

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:

- **Cost Reductions:** Synergies lead to less duplication and lower expenses.
- **Accelerated Innovation:** Savings from cost reductions free up resources for innovation.

- **Reuse and Sharing:** Creating synergistic components enables more possibilities for reuse.
- **Increased Efficiency:** Well-executed transformations result in greater efficiencies and lower unit costs.

However, striving for synergies and efficiency presents challenges:

- **Up-Front Investment:** Significant initial investment is required to realize benefits, which carries high risks.
- **Performance Pressure:** Teams must deliver excellent short-term results while undergoing significant transformations.
- **Temporary Productivity Drops:** Balancing transformation activities with regular work can temporarily reduce productivity.
- **Increased Complexity:** 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.

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 shire777 from pixabay

Due to historical and strategic reasons, 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:

- **Higher Flexibility:** Units can develop independently, addressing specific needs without synchronizing with other units.
- **Faster Time-to-Market:** Fewer dependencies enable marketplaces to rapidly change and evolve their products for local needs.
- **Innovation:** Opportunities to quickly explore ideas in smaller contexts.

However, loose coupling also presents several challenges:

- **Duplication of Effort:** While local market needs differ, there is often significant overlap in product features and technology, leading to redundant efforts as each marketplace creates solutions for the same problems.
- **Increased Accidental Diversity:** Limited synchronization may result in significantly different design and technology choices for the same problem, making it challenging to consolidate solutions, move personnel between teams, or benefit from economies of scale.
- **Limited Central Control:** Fewer dependencies and varying goals make it more difficult to implement changes across the organization.

From an architectural perspective, loose coupling presents an interesting challenge as it often leads to a conflict between global alignment and control and local autonomy.

2.6: Questions to Consider

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

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

3: Evolution of Architecture: Embracing Adaptability, Scalability, and Data-Driven Decisions



image by bluehouse skis from pixabay

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

KEY POINTS:

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

Grounded Architecture emerged as a necessity in response to our **intricate and multifaceted challenges**. Grounded Architecture 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** architectural practice.

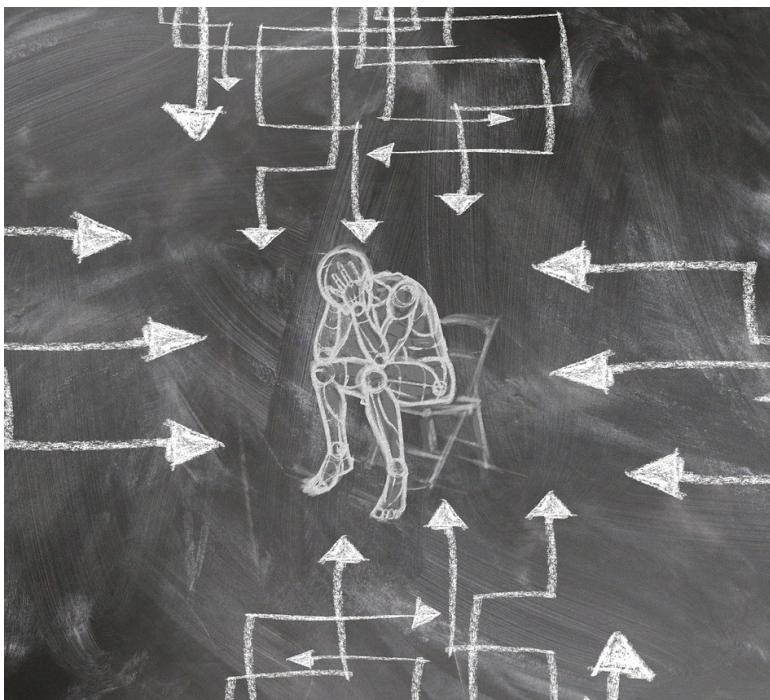


image by gerd altmann from pixabay

3.1: Goals

Here's a detailed breakdown of the goals we set to revamp our approach.

3.1.1: Goal 1: Executing At Scale

Our organization comprises hundreds of teams and thousands of projects, each with unique complexity and requirements. Traditional, one-size-fits-all approaches to architecture practice cannot keep up. We needed a system that could support this vast and varied ecosystem. Grounded Architecture was designed to provide the scalability necessary to **handle such diversity at scale**, ensuring that teams and projects received the support they needed without being constrained by the limitations of processes.

3.1.2: Goal 2: Adaptivity

In our dynamic environment, 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.

3.1.3: 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 advanced tools and mechanisms to support data-driven decision-making. By leveraging data and analytics, we can move away from subjective opinions and towards objective, evidence-based decisions. This approach enhances the quality of our decisions and ensures that they are consistent and aligned with our organizational goals.

3.1.4: 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.

3.1.5: 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 is designed to facilitate continuous learning and growth. It supports the rapid adoption of new technologies and encourages **ongoing education and training**. By keeping the organization up-to-date with the latest developments, Grounded Architecture aims to ensure we always have the best tools and knowledge to drive innovation and improvement.

Grounded Architecture aims to transform a typically cumbersome, manual-process-driven practice into a streamlined, adaptive, and learning-focused powerhouse.

3.2: Questions to Consider

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

Part I: Grounded Architecture

4: Grounded Architecture: Introduction



image by ichigo121212 from pixabay

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

KEY POINTS:

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

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

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

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

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

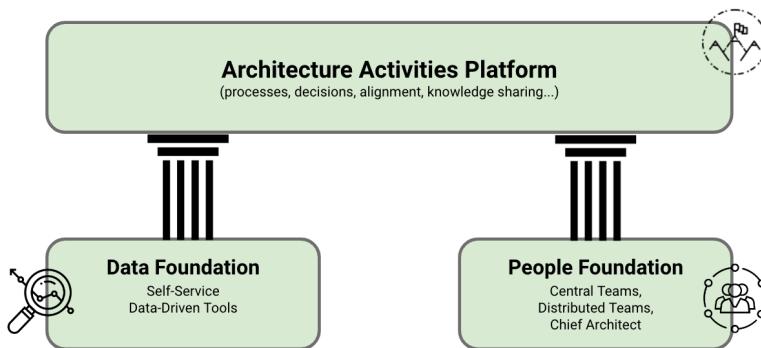


Figure 1: The structure of Grounded Architecture.

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

The *People Foundation* is a **network of people** doing architecture across the organization. This Pillar is crucial to ensure that the architecture function has any **tangible impact**. The [People Foundation section](#) provides more details.

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

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

Now that we've got the Grounded Architecture tour out of the way, let's dig into the specifics.

5: Data Foundation

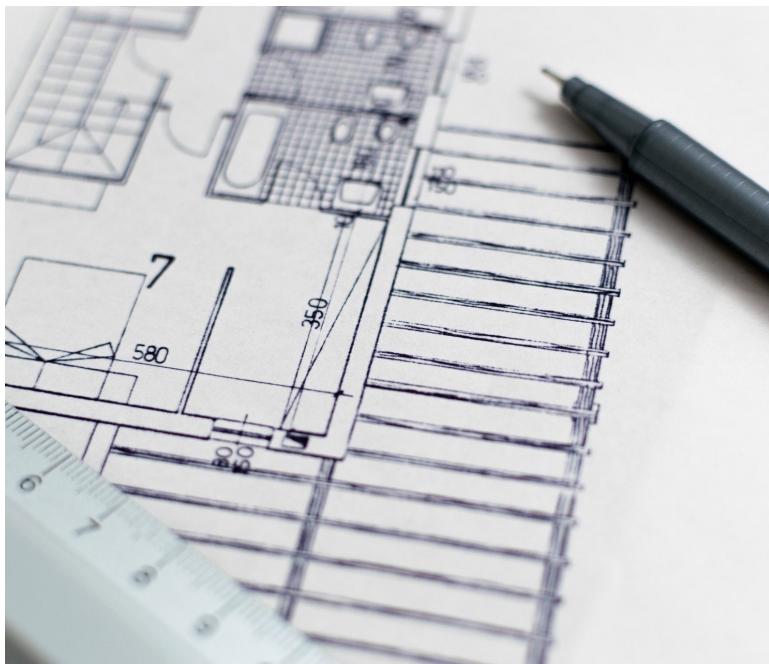


image by lorenzo cafaro from pixabay

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

KEY POINTS:

- The architecture Data Foundation serves as a medium to create a complete, up-to-date picture of critical elements of the technology landscapes of big organizations.
- The Data Foundation provides an architecture-centric view of data about a technology landscape based on source code analyses, public cloud billing reports, vibrancy reports, or incident tickets.
- To facilitate the creation of a Data Foundation, I have been working on creating open-source tools that can help obtain valuable architectural insights from data sources, such as source code repositories. Check out open-source [architecture dashboard examples](#)¹ and [Sokrates](#)².

“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 architectural functions, I strongly (aka obsessively) emphasized data. Consequently, one of the first steps I make in any architecture practice is to create an architecture Data Foundation to get a complete, up-to-date picture of critical elements of an organization’s technology landscapes (Figure 1). Manual documentation does not scale, and relying on data ensures the reliability and scalability of decision-making. In the past several years, I have also been working on creating open-source tools, such

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

²<https://sokrates.dev>

as [Sokrates](#)³, that can help obtain valuable architectural insights from data sources, such as source code repositories or public cloud billing reports.

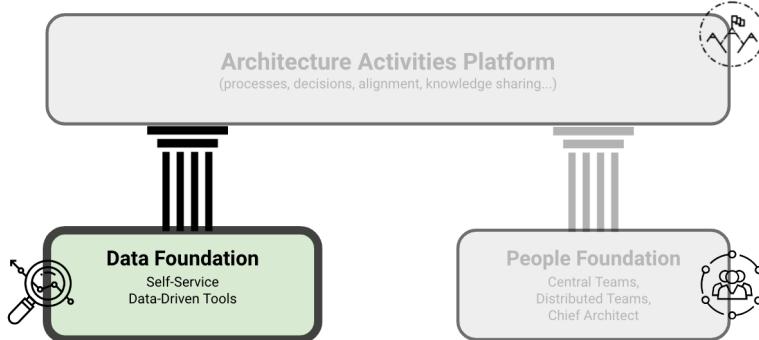


Figure 1: The structure of Grounded Architecture: The Data Foundation.

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

³<https://sokrates.dev>

5.1: Examples of Data Foundation Tools

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

- **Source code** 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, we can identify critical areas of improvement, understand team dynamics, and ensure code quality.
- **Public cloud billing reports** provide an overview and trends about used cloud services, regions, and budgets. Monitoring billing reports helps 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 reveals trends, common issues, and dependencies among incidents, helping in proactive problem management and improving system reliability.
- **Key business metrics**, like vibrancy, which can show user activity on our systems. Tracking these metrics helps in assessing the health of the business, understanding user behavior, and guiding strategic decisions to enhance user experience.
- **Messaging and collaboration platforms (such as Slack) activity reports**, which can help understand discussion topics and team interactions. Analyzing these reports helps in understanding collaboration patterns, identifying key discussion areas, and improving team communication and productivity.

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

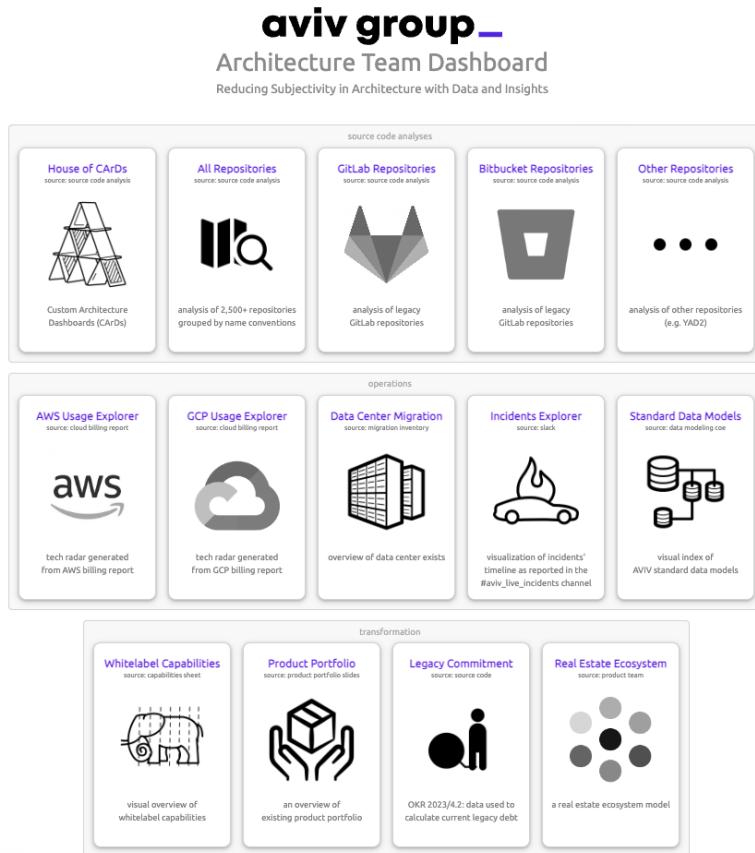


Figure 2: A screenshot of the start page of the architecture data dashboard we've built and used at AVIV Group.

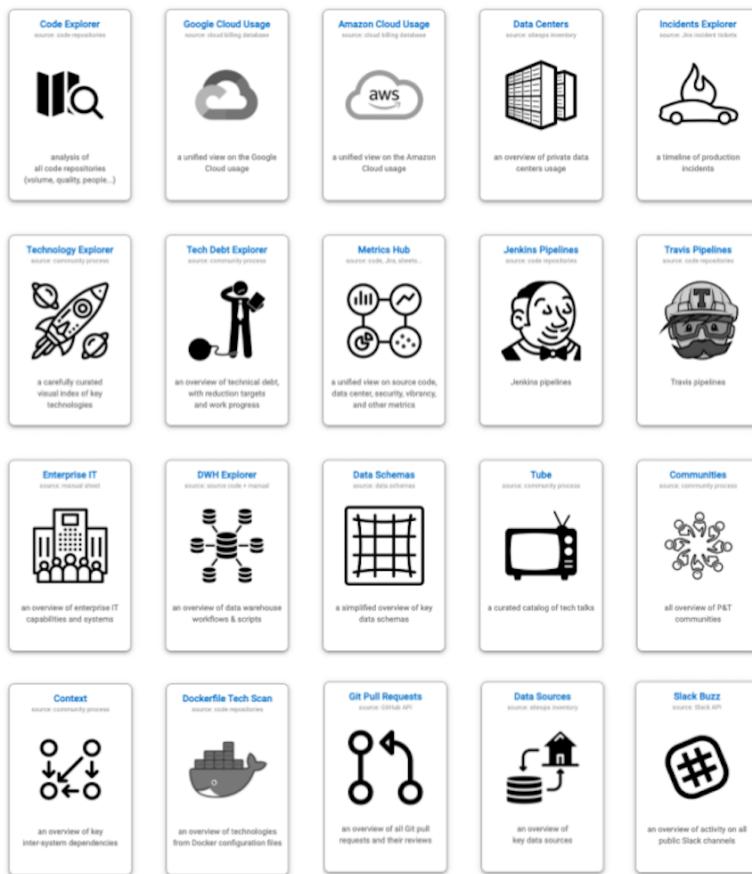


Figure 3: A screenshot of the start page of the architecture data dashboard we've built and used at eBay Classifieds.

5.1.1: Example 1: Source Code and Commit History

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. To help dig up this treasure without getting your hands too dirty, I've developed and actively maintain a project called [Sokrates](#)⁴.

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 both CTO-level strategy powwows and developer-level code critiques.

For a more entertaining look at what Sokrates can do, check out the [Sokrates examples](#)⁵. Here are some blockbusters:

- [Apache Software Foundation Repositories](#)⁶: An epic saga of over 1,000 repositories with more than 180 million lines of code, 22,000 contributors, and 2.4 million commits.
- [Facebook/Meta OSS Repositories](#)⁷: A thriller with 800 repositories, 120 million lines of code, 20,000 contributors, and over 2 million commits.
- [Microsoft OSS Repositories](#)⁸: 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](#)⁹: 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](#)¹⁰: 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](#)¹¹: A thriller with over 2,700

⁴<https://sokrates.dev>

⁵<https://www.sokrates.dev/>

⁶https://d3axxy9bcycpv7.cloudfront.net/asf/_sokrates_landscape/index.html

⁷https://d3axxy9bcycpv7.cloudfront.net/meta/_sokrates_landscape/index.html

⁸https://d3axxy9bcycpv7.cloudfront.net/microsoft/_sokrates_landscape/index.html

⁹https://d3axxy9bcycpv7.cloudfront.net/google/_sokrates_landscape/index.html

¹⁰https://d3axxy9bcycpv7.cloudfront.net/asf/_sokrates_landscape/index.html

¹¹https://d3axxy9bcycpv7.cloudfront.net/amzn/_sokrates_landscape/index.html

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

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

Amazon Web Services (AWS)¹², Google Cloud Platform (GCP)¹³, Microsoft Azure¹⁴, 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.

¹²<https://aws.amazon.com>

¹³<https://cloud.google.com/>

¹⁴<https://azure.microsoft.com/>

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

The screenshot displays a heatmap representing Google Cloud Platform usage data. The top navigation bar includes checkboxes for 'Total', 'Per Resource Family', 'Per Budget', and 'Per Service'. Below this is a header row with icons and labels for 'Total', 'Compute', 'Network', 'Storage', 'Application Services', 'License', 'Compute Engine', 'Kubernetes Engine', 'Cloud Storage', 'Cloud CDN', 'Cloud DNS', 'BigQuery', and 'BigQuery Reservation API'. The main area is a grid where each cell contains a dollar sign (\$) followed by three asterisks (***) to indicate high-level cost data. A vertical sidebar on the left is labeled 'SUM' and features a series of colored bars (blue, red, green, yellow, purple) with checkmarks, likely representing different resource categories or filters applied to the data.

Figure 4: An example of a cloud usage explorer.

5.1.3: Example 3: Financial and Vibrancy Data

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**.

These finance sleuths 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. By linking 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!

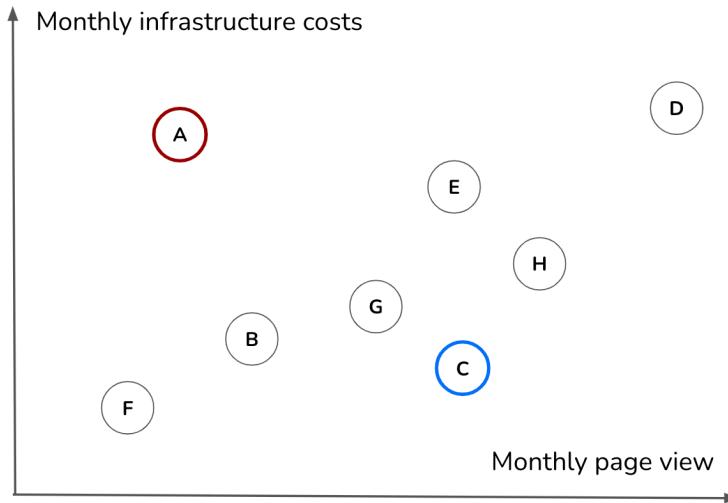


Figure 5: Combining data from a different source (e.g., cloud billing reports and vibrancy or revenue can lead to new insights (e.g., identifying inefficiencies in the application portfolio).

5.1.4: Example 4: Data-Driven Capability Map

Capability models¹⁵ are frequently associated with Enterprise Architecture, providing a structured approach to organizing and managing an organization's capabilities. Traditionally, these capability maps are maintained manually, which can be time-consuming and prone to inaccuracies. A data-driven version of the capability map enhances this approach by integrating real-time data sources, offering a more dynamic and accurate representation of capabilities. A data-driven version of the capability map enhances this approach by integrating real-time data sources, making the capability map **dynamic, alive**, and significantly more useful.

¹⁵<https://pubs.opengroup.org/togaf-standard/business-architecture/business-capabilities.html>

I have created several versions of data-driven capability maps, where a capability map includes capability data cards that consolidate various data types relevant to each capability. We then automatically generated sites and visuals from these data for self-service use within the organization. These cards provide a comprehensive view of the current state and performance of each capability, integrating multiple data sources:

1. Documentation Data:

- **Links to Key Documents:** Direct access to important documents related to the capability.
- **Automated Summaries:** Using generative AI to automatically summarize key documents, making it easier to quickly understand the essential points and status.

2. Technical Implementation Evidence: Source Code Repositories:**

These repositories link to and analyze all source code related to the capability, providing insight into the technical implementation.

- **Cloud Accounts and Billing Reports:** Detailed analyses of cloud accounts, including costs, services used, and geographical distribution (regions).
- **Infrastructure Costs and Analyses:** Detailed breakdown of the capability's associated infrastructure costs.
- **Legacy Assets List:** This is an Inventory of legacy systems and components that are part of or impact the capability.
- **Technical Debt Inventory:** Identification and assessment of technical debt associated with the capability.

3. Planning Evidence:

- **Detailed Planning Data:** Information from planning systems like Jira, including detailed task breakdowns and timelines.

- **Effort Estimates:** Projections of the efforts required to develop or maintain the capability.

4. Execution Data:

- **Product and Business Evidence:**
 - **Documentation of Demos:** Links to slides, videos, and other demo materials.
 - **Product Analytics Data:** Metrics such as user engagement (vibrancy), conversion rates, and other relevant analytics.
- **HR Evidence:**
 - **People and Teams:** Information about the personnel and teams working on the capability.
 - **Time Tracking Data:** Detailed reports on the time spent on the capability by different team members.

Real-time data integration allows for creating dynamic and aggregated views across capabilities. These views can be grouped by criteria, such as domain programs or themes, providing a broader perspective on how capabilities interrelate and contribute to strategic objectives. Examples include:

- **Domain Programs:** Grouping capabilities by specific business domains (e.g., finance, marketing, operations) to understand domain-specific strengths and weaknesses.
- **Themes:** Aggregating capabilities around strategic themes or initiatives (e.g., digital transformation, customer experience improvement) to track progress and resource allocation.

Key benefits of a data-driven capability map include:

1. **Enhanced Decision-Making:** Real-time data provides a current and accurate picture of capabilities, enabling better-informed strategic decisions.

2. **Increased Transparency:** Detailed evidence and summaries help us understand the status and needs of each capability.
3. **Improved Efficiency:** Automated data aggregation and reporting reduce manual effort and speed up the analysis process.
4. **Strategic Alignment:** Aggregated views help ensure that capabilities are aligned with broader organizational goals and initiatives.
5. **Resource Optimization:** Detailed cost and effort data help optimize resource allocation and manage budgets effectively.

By leveraging a data-driven capability map, organizations can achieve a more dynamic, transparent, and efficient approach to managing their capabilities, leading to improved strategic outcomes and operational performance.

5.2: Requirements For A Data Foundation

A Data Foundation should be a central place with **authoritative, relevant, and curated data** about the organizational technology landscape. Technically, you can implement Data Foundation tools like those discussed in the previous section, using simple tools like Google Drive, with documents organized in folders or as an internal website. I recommend investing some effort in creating better infrastructure and user experience, as it can enable more people to access and benefit from data. 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 Data Foundation, with papers on the wall, in Google Drive, in Confluence, or with a nicely designed internal website, I have identified the following requirements that a Data Foundation needs to have:

- **It is the single point of truth** for all relevant architectural data. People should be able to go to one place and get the most relevant data.
- **It is curated for quality** so people can trust the data. Simply dumping data into one place will not help. You need to own curation to ensure that data are correct. You also should provide links to data sources so people can verify the facts.
- **It is curated for usability** so people stay focused on valuable details. You must filter out useless or less relevant details, focusing on the essence. Investing in the UX design of documents or tools you create helps.
- **It is kept up to date**, ideally in an automated fashion (or in a semi-automated repeatable way).

- **It is accessible to the whole organization.** I genuinely believe that when you give employees access to information generally reserved for specialists, architects, or “higher levels,” they get more done independently. They can work faster without stopping to ask for information and approval. And they make better decisions without needing input from architects or the top.
- **It is used in decision-making.** Having nicely curated and valuable data has zero value if you cannot ensure that such data inform vital decisions.

My approach to building the Data Foundation is like creating 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 Data Foundation 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 Data Foundation 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 Data Foundation

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

- **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](#)¹⁶. 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 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.** Sharing fewer data with everyone is like handing out fewer candies at Halloween—easier and less chaotic. Keep it simple, avoid complex authorization mechanisms, and you'll have fewer data goblins to manage.
- **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 the Data Foundation.
- **Use simple and easy-to-maintain infrastructures.** For example, I publish the results of Sokrates analyses and other

¹⁶<https://sokrates.dev>

data tools as static resources on our enterprise GitHub pages. Avoid the headache of complex databases and backend software. In the [Architecture Dashboard Examples repository¹⁷](#), you'll find the source code for building the Architecture Data Dashboard. The dashboard is a simple static website generated from JSON files and [published via GitHub pages¹⁸](#).

These tips might just 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 along the way.

¹⁷<https://github.com/zeljkoobrenovic/grounded-architecture-dashboard-examples>

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

5.4: Using Architecture Data Foundation

The Data Foundation 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 code 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 dependen-

cies. 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: Appendix: Examples of Insights From Source Code Analyses

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

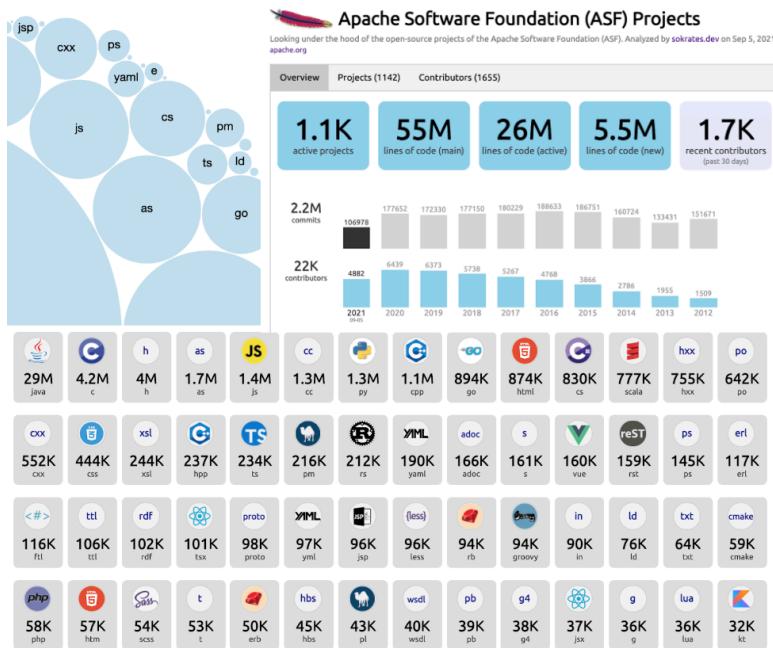


Figure 6: Sokrates can instantly create a helicopter view of the technology landscape, programming languages, active contributors, and commit trends.

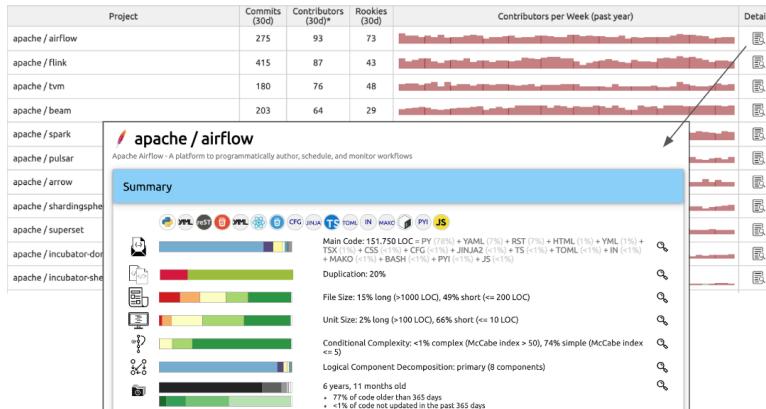


Figure 7: Sokrates can show detailed code and contributors' trends per repository, enabling zooming in each repository up to the code level.

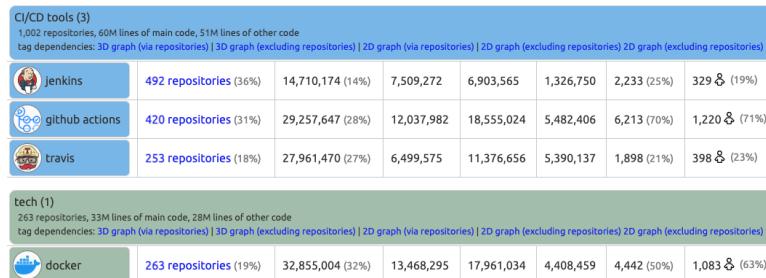


Figure 8: Sokrates can create a tech radar by tagging projects with identified technologies.

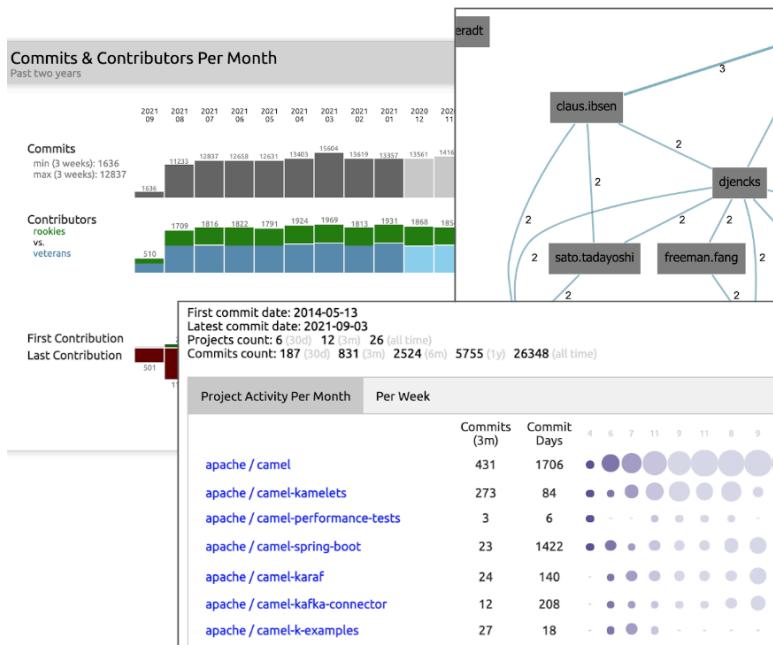


Figure 9: Sokrates can show contributor trends, distribution of “veterans” and “rookies,” and dependencies between people and repositories, enabling zooming in into patterns of the contribution of individual contributors.

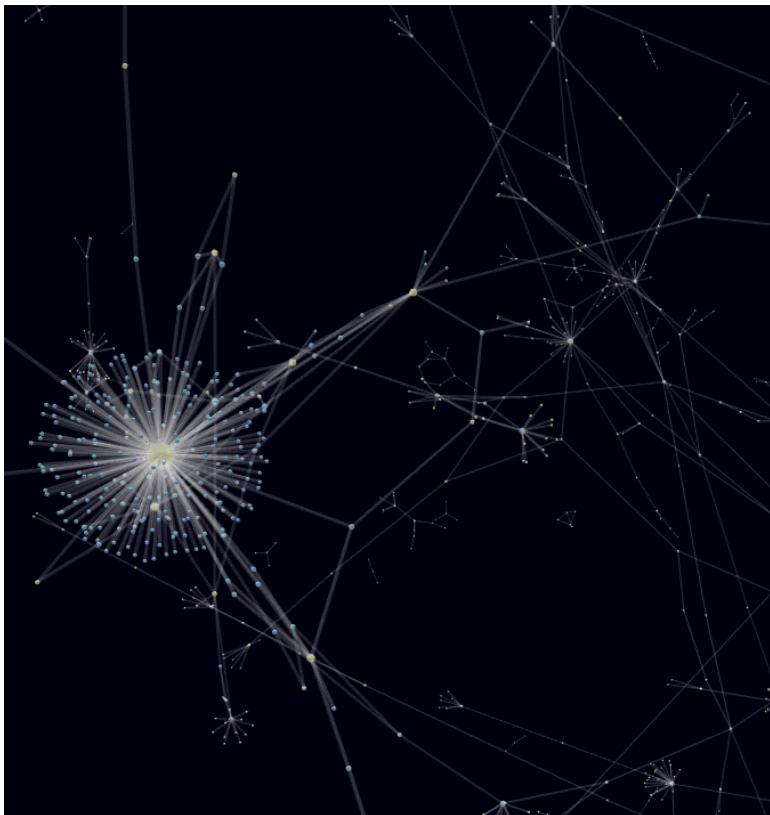


Figure 10: Sokrates can reveal the team topologies by plotting 2D and 3D graphs of dependencies that people create through working on the same repositories in the same period.

5.6: Questions to Consider

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

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

6: People Foundation



image by mostafa meraji from pixabay

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

KEY POINTS:

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

As the wise Gregor Hohpe noted, transforming an organization isn't about solving mathematical equations. No, it's about moving people. Developing the architecture function requires competent, empowered, and motivated architects. This makes **a strong network of people doing architecture** crucial for any real impact. In simpler terms, **Strong Architecture = Strong Architects**.

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 what you can do is to carefully organize, empower, and leverage this scarce talent (Figure 1).

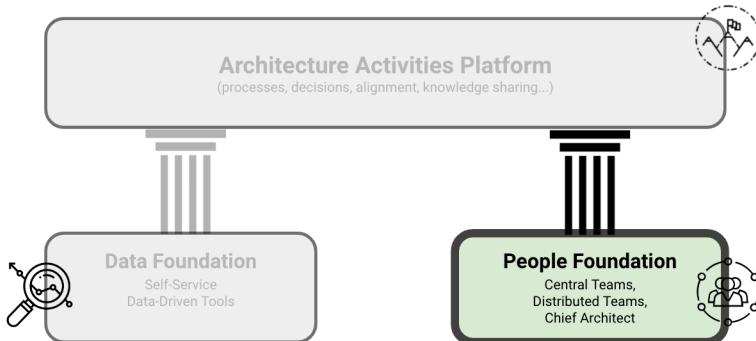


Figure 1: *The structure of Grounded Architecture: The People Foundation.*

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.

6.1: Background: Central vs. Federated Architecture Function

IT architecture practices generally follow one of two fundamental models: central or federated (Figure 2, [McKinsey 2022¹](#)).

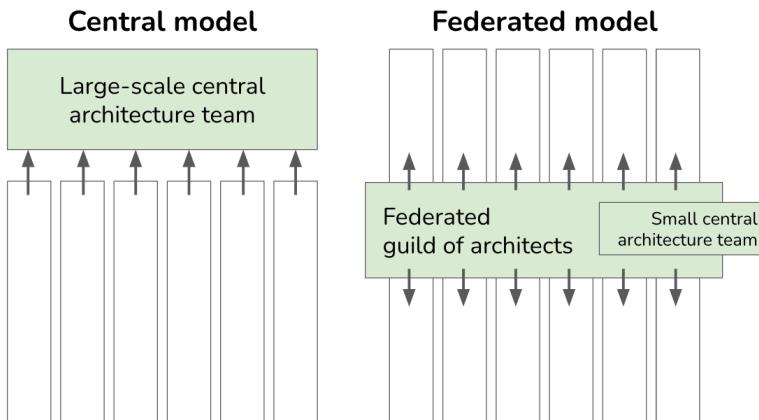


Figure 2: Central vs. Federated Architecture Function..

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.

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

The **federated model** is a favorite among **cross-functional DevOps cultures**, integrating infrastructure, operations, and security into their brews. Architects in this model focus on **facilitation and enablement**, not just making sure no one spills the beans. Modern agile organizations love the federated model. It keeps architects close to the action, ensuring they're evaluated based on the success of the products they support. This focus on performance and reducing complexity is the secret recipe for a well-brewed architecture.

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. It's like having a superhero headquarters where everyone knows the plan to save the day.

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 the value of dedicated roles in central architecture teams. These roles, in addition to the usual suspects doing architecture work, are instrumental in covering these crucial yet often overlooked responsibilities:

- **Build and maintain the architecture Data Foundation.** 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

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

I'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?

Our guilds or virtual teams 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.

When we have an overabundance of guilds and teams, we split our architects into different cliques:

- **General or core teams:** These folks tackle a wide range of general architectural topics.
- **Specialist teams:** Focused on specific parts of the tech stack, like native mobile apps, web frontends, or public cloud.
- **Strategic initiatives teams:** Working on the big picture stuff like data strategy, public cloud strategy, transactions, or verticalization.

Connecting the central and distributed teams is essential. In most places I've worked, these gatherings follow a well-trodden path:

- **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 as many people as possible 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!

6.2.3: Embracing Diverse Team Structures

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.

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.

6.3: Building People Foundation

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

- Before making grandiose plans for reorganizations, **connect with the people already doing architectural work** in an organization. Think of it like forming a superhero team: bring together all the critical tech leaders, regardless of their position and title. You never know when you’ll need an architectural Batman or Wonder Woman. Being well-connected to these folks will be crucial in any architecture organization, so this effort is like adding a secret weapon to your utility belt.
- If creating a virtual team is 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 stakeholders** early to gain broader support for building architecture teams and guilds. Imagine it as building alliances in a grand strategy game; you need their support to conquer the world of architecture. Being well-connected to these stakeholders is crucial, so think of it as adding allies to your quest.
- **Avoid hiring a digital hitman**². 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.

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



image by chantellev from pixabay

6.4: Questions to Consider

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

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

7: Architecture Activities Platform

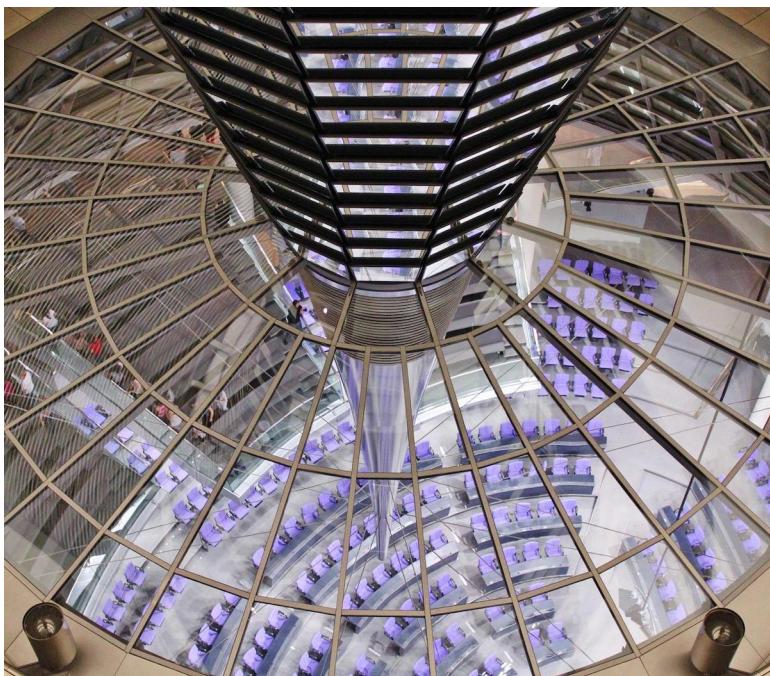


image by anja from pixabay

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

KEY POINTS:

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

Each organization will have different architectural needs and contexts. When forming architecture functions, I use as a starting point these [two pieces of advice from Gregor Hohpe](#)¹:

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

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

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

stakeholders. This collaborative approach is essential to create a sustainable and practical architectural function. It's like establishing the house rules for game night; everyone's input is valued and necessary to keep things fun and functional.

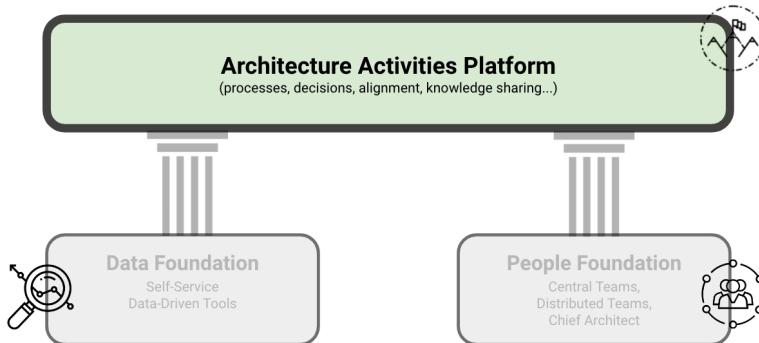


Figure 1: The structure of Grounded Architecture: Architecture Activities Platform.

The Architecture Activities Platform (Figure 1) is a set of processes and agreements that allows architects to do everything architecture practice typically does. It leverages Data and People Foundations to develop a data-informed, organization-wide impact. Data and People Foundations provide a basis for data-informed decision-making well embedded in the organization. It's about instilling confidence and trust in your decision-making process.

7.1: Examples of Architecture Activities

To provide a clearer understanding of what I mean by an Architecture Activities Platform, here are detailed examples of the activities I have been engaged in with architects.

Designing Mechanisms for Teams to Make Better Decisions - This involves creating global decision-support frameworks such as advisory forums, which facilitate 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 entails integrating into key team activities, aligning architectural work with team rituals to provide timely support. We assist 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 work 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 provide data and insights about the legacy landscape, identifying problematic areas such as frequently changed, low-quality, untested legacy code. We help 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 perform SWOT (Strengths, Weaknesses, Opportunities, Threats) analyses and other assessments. These analyses help in creating comprehensive plans and roadmaps for improvement.

Standardizing Processes and Documentation - We define 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 provide analyses, recommendations, and integration planning for mergers and acquisitions. My support ensures that architectural considerations are well-integrated into M&A activities, facilitating smoother transitions and integrations.

Defining Key Technology Strategies - We contribute 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 work on creating a sustainable organizational setting that aligns with the overarching technology strategies. This involves setting a clear vision and direction for the technology landscape within the organization.

These activities collectively form an Architecture Activities Platform, enabling a structured and strategic approach to architectural practices 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. The operating model emphasizes a collaborative and supportive approach. Architects empower development teams to make most decisions while ensuring strategic alignment and minimal compatibility. Architects 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 Model

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

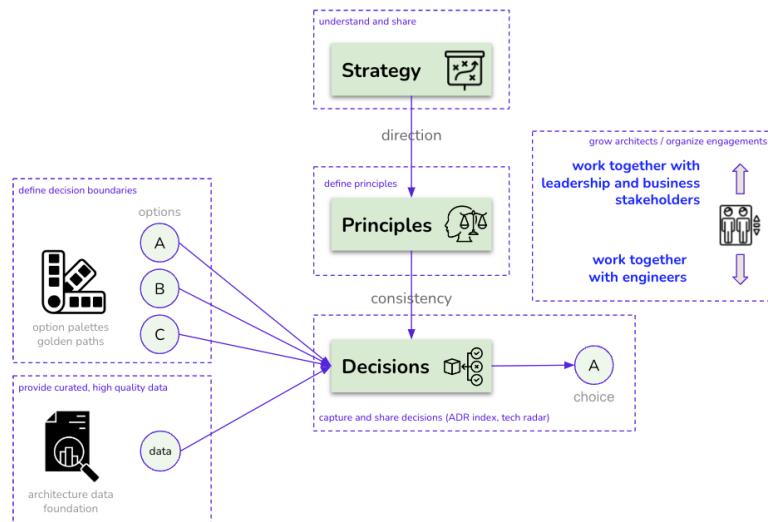


Figure 2: A common operating model I typically use for Grounded Architecture activities.

Here are the key characteristics of this operating model.

Engagement mindset:

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

Contributions of architects:

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

- Share and generalize lessons learned via a [People Foundation](#).

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 and do not align with anyone, do not get feedback from anyone, and do whatever they want. Teams must complement autonomy with high transparency and proactivity in alignment with other groups.

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

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



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

7.2.3: 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](#)², “Act in Netflix’s best interests”, I frequently argued that architecture decision policy could similarly be summarized in six words:

²<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, get promoted): Do not unnecessarily create exotic islands with few experts in technologies not supported or widely used in the organization. People cannot get help or move across the organization as their expertise may be useless outside the team.
- **Ease of training and onboarding** of internal and external developers: Using conventional technologies, supported with external learning resources (books, tutorials, StackOverflow) significantly helps find and grow experts.
- **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. It cannot be achieved 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 resonates well. It encourages people 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 implementation³](#). Golden Paths can be described as “**opinionated and supported**” routes developers can follow to build systems efficiently and effectively.

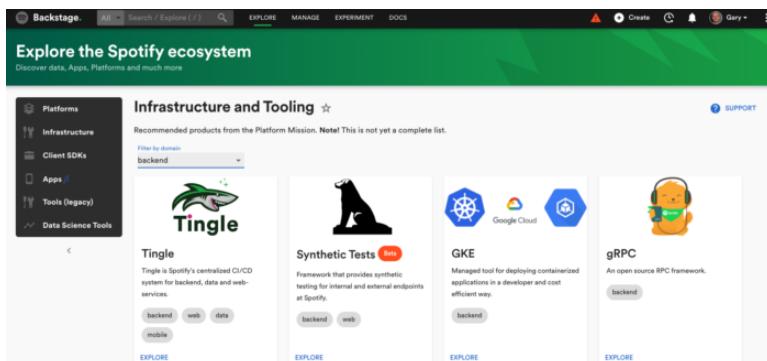


image by engineering.atspotify.com

Golden paths provide a solid **foundation for aligning** architecture activities, serving as a common target of work for Guilds and

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

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 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: Setting Boundaries

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



image by gordon johnson from pixabay

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

conflict-resolution mechanisms. Essentially, ROE sets the stage for what's expected and what's not, ensuring everyone plays nicely and fairly.

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

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

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

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

7.4: Questions to Consider

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

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

8: Transforming Organizations with Grounded Architecture



image by istock

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

KEY POINTS:

- When a Grounded Architecture structure is in place, it can 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 structure is in place, it can positively transform an organization's functioning. These categories of impact, aligned with **defined goals**, are:

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

8.1: 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.

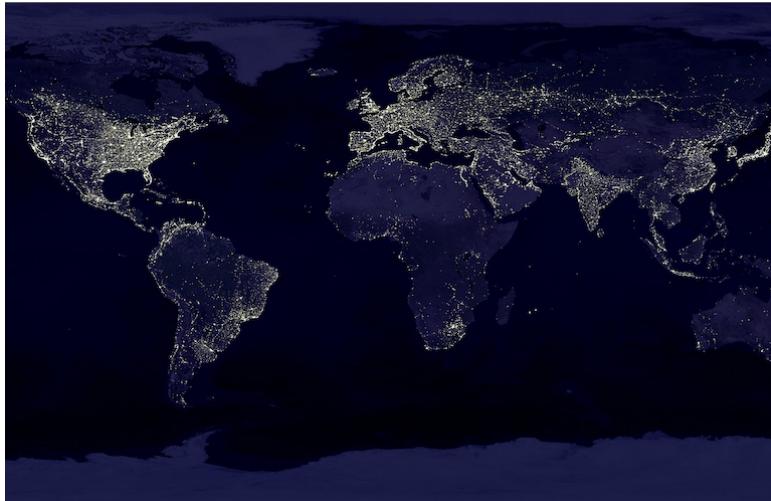


image by wikiimages from pixabay

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

8.1.1: Data Foundation

The **Data Foundation** 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 can be facilitated through internal websites or portals where data tools and insights are readily available.

- **Elimination of Meetings:** We significantly reduced the need for numerous information-sharing or data-gathering meetings using automated data tools (every time we added a new data app, I got a few hours back in my calendar and had fewer Slack messages). This reduction saves time and ensures that data is consistently 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.

8.1.2: People Foundation

The **People Foundation** 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.

8.1.3: Architecture Activities Platform

The **Architecture Activities Platform** promotes scalability by:

- **Distributed Decision-Making:** Encouraging organizational decision-making prevents bottlenecks associated with centralized decision-making processes. This 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.

8.2: Adaptivity

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



image by francis ray from pixabay

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

8.2.1: Data Foundation

The Data Foundation can support adaptability by:

- **Automation and Extensibility:** We could swiftly extend and reconfigure our data infrastructure by implementing a highly automated Data Foundation to accommodate changes. For example, this could involve integrating new source code

repositories and data sources following acquisitions or mergers. 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:** A robust Data Foundation adapts its views to the changing realities of different parts of the organization, ensuring that data insights are always aligned with current operational contexts.

8.2.2: People Foundation

The **People Foundation** can support adaptability by:

- **Capacity Redistribution:** The People Foundation, through a central team, can alleviate the temporary capacity shortages experienced by distributed teams. This ensures that architectural functions continue to operate smoothly even during periods of high demand or limited resources.
- **Decentralized Support:** By maintaining a network of well-connected architects, the architecture function 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.

8.2.3: Architecture Activities Platform

The **Architecture Activities Platform** can support adaptability by:

- **Flexible Decision-Making:** The Architecture Activities Platform supports adaptivity by promoting a flexible setting

and distributing decision-making authority throughout the company. This distribution helps prevent the architecture function 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. This includes 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 the architectural function's flexibility and adaptivity. It ensured we could respond promptly and effectively to new challenges and opportunities.

8.3: Improving the Quality of Decision-Making with Data

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

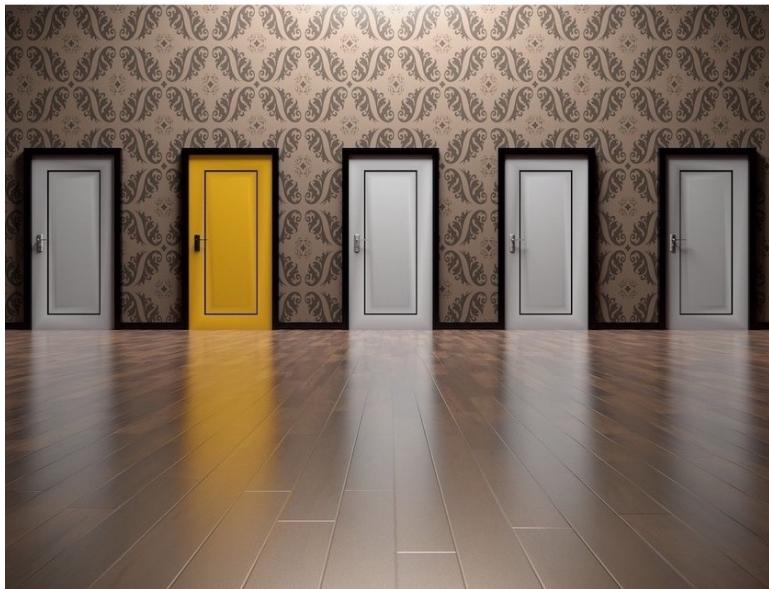


image by arek socha from pixabay

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.

8.3.1: Data Foundation

The **Data Foundation** ensures the availability and readiness of data needed for decisions, fueling data-informed discussions and decision-making via:

- **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 advanced 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.

8.3.2: People Foundation

Data is not enough; you need to make this data available to and used by critical people. The **People Foundation** ensures people are available and well-connected to share information and make decisions via:

- **Expert Network:** Establish and maintain a network of knowledgeable 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.

8.3.3: Architecture Activities Platform

The **Architecture Activities Platform** 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 objectively 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.

8.4: Maximizing Organizational Alignment

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

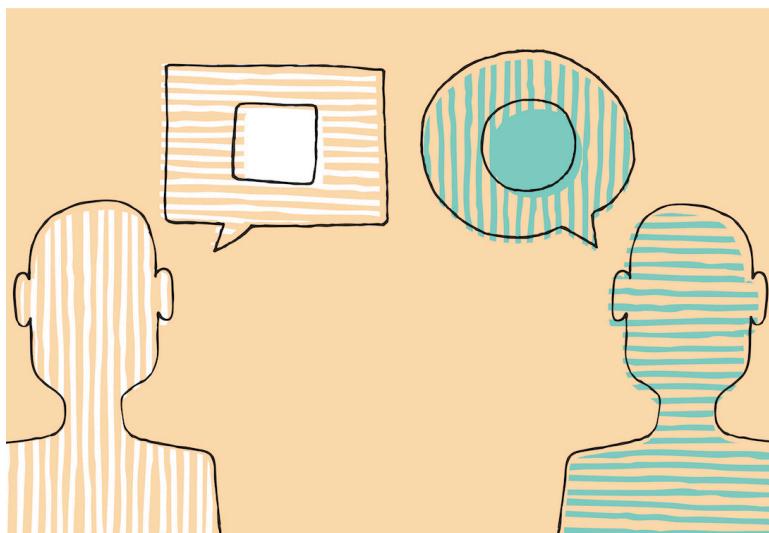


image by istock

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.

8.4.1: The Data Foundation

The **Data Foundation** improves organization alignment by creating transparency. By establishing a robust data foundation,

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.

8.4.2: The People Foundation

The **People Foundation** improves organization alignment by **facilitating collaboration**. The People Foundation 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.

8.4.3: The Architecture Activities Platform

The **Architecture Activities Platform** 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, the platform ensures 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 function 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.

8.5: Maximizing Organizational Learning

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

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



image by istock

The Grounded Architecture structure offers comprehensive support through its Data Foundation, People Foundation, and Architecture Activities Platform to help organizations learn quickly, stay up-to-date with emerging technologies, and recommend technology upgrades.

8.5.1: Data Foundation

The Data Foundation plays a pivotal role in accelerating the learning and adoption of new technologies. Here's how:

- **Facilitating Exploration and Reflection:** The Data Foundation 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.

8.5.2: People Foundation

The People Foundation enhances 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 are 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:** The People Foundation ensures 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.

8.5.3: Architecture Activities Platform

The Architecture Activities Platform 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 the Data Foundation, People Foundation, and Architecture Activities Platform, the Grounded Architecture structure

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.

8.6: Questions to Consider

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

- *How effective is your organization's current architectural function at scale? How valuable could principles of Grounded Architecture be 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 architectural function? Could the Data and People Foundations principles be utilized to improve alignment?*
- *What strategies does your organization currently have to foster organizational learning? How could the methods described in the Grounded Architecture model enhance this?*
- *How quickly can your organization adopt and utilize new technologies? How could your architecture practice accelerate this process?*
- *Consider the adaptivity of your organization's architectural function. How could your architecture practice improve it?*
- *Reflecting on the value of the “Data Foundation” concept, how effectively is your organization tracking changes or supporting what-if scenarios analysis?*
- *What role do most senior architects play in your organization? Could their time be better utilized on strategic initiatives?*
- *How sustainable is the architectural function in your organization in the absence of a strong central team? Could implementing a Data Foundation and well-connected architects help mitigate this?*

Part II: Being Architect

9: Being Architect: Introduction



image by borko manigoda from pixabay

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

In this part of the book, I introduce several perspectives on what it means to be an architect in practice:

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

10: Architects as Superglue



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

KEY POINTS:

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

I believe architects in IT organizations should develop into “superglue.” I borrowed this concept from [Adam Bar-Niv and Amir Shenhav from Intel](#)¹, who suggested that instead of needing superheroes, we need “superglue” architects—the people who hold architecture, technical details, business needs, and people together across large organizations or complex projects. More recently, Tanya Reilly presented a [similar view](#)² concerning software engineering roles.

Superglue architects act as the **organizational connective tissue**, linking the “**business wheelhouse**” with the “**engine room**.” While technical prowess is essential, their unique strength lies in their ability to relate, or glue, technical issues with business and broader organizational matters.

Based on discussions with engineering leaders, engineers, and architects, Figure 1 illustrates the “superglue” metaphor for architects.

Architects must build strong relationships with developer teams, local business stakeholders, and various functions. Simultaneously,

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

²<https://noidea.dog/glue>

they need to be well-connected with broader internal communities. External visibility is also crucial, allowing architects to bring fresh ideas into the organization and promote the organization to the outside world.

In essence, while superheroes might be great for saving the day, it's the superglue architects who keep everything together, ensuring that the whole machine runs smoothly. So, forget about donning a cape—grab a bottle of superglue and start connecting the dots!

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

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

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

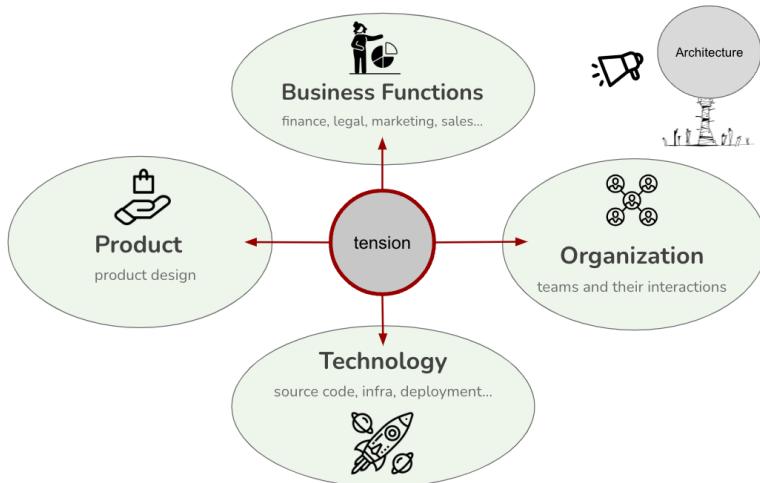


Figure 2: The tensions between technology, product, organization, and business functions.

The main problem with this tension is that it can significantly slow things down due to **miscommunication** or other **misalignments**, lead to costly and **detrimental decisions** due to lack of information, introduce overwhelming and **unnecessary complexity**, and cause critical **missed opportunities**. This is a pressing issue that needs immediate attention. Too frequently, architecture sits on the sidelines, shouting principles and abstract ideals that everyone ignores.

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

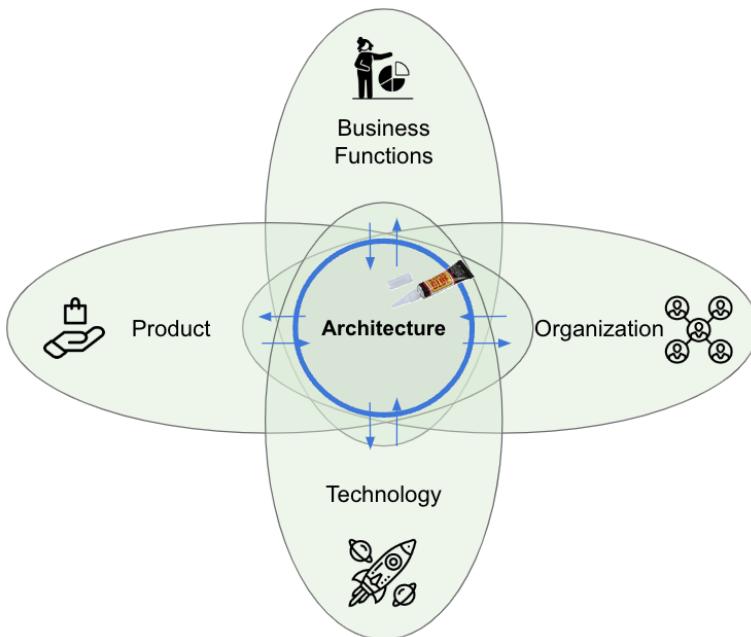


Figure 3: Architects should be in the middle of reducing tensions between technology, product, organization, and business functions.

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

- **Building the wrong products:** If technology implementation makes incorrect assumptions not aligned with product

requirements, you might end up with a toaster when you really needed a coffee maker.

- **Wrong prioritization of activities:** Without clear business and product metrics, we may build “interesting” products that do not create value for our customers and business. Picture making an app that tracks how often your cat blinks—fascinating, but not exactly a bestseller.
- **Unexpected delivery delays:** Underestimating complexity, effort, and dependencies can lead to delays that make your project feel like it’s stuck in a time loop.
- **Duplication of effort:** Without business or product harmonization needed to facilitate building shared components, you end up reinventing the wheel so often you could start a wheel shop.
- **Building too complex products:** Creating an overly complex, configurable system to address all possible cases can lead to a labyrinthine solution when a simpler, more harmonized approach would have sufficed. It’s like using a Swiss Army knife when you just needed a spoon.
- **Overengineering:** A lack of pushback to simplify products and a lack of understanding of technology can lead to using a complex and expensive messaging middleware capable of handling millions of messages per hour for a use case where you only have a few thousand messages per day. It’s like buying a monster truck to drive to the grocery store.
- **Building too simple, unscalable products:** Assuming we will simplify and harmonize our processes, but in reality, needing to keep essential complexity and support many variations, can result in a system that’s as robust as a house of cards.
- **Building low-quality products:** Unnecessary complexity and a lack of critical knowledge and expertise in the organization can lead to products that fall apart faster than a dollar-store umbrella in a hurricane.

- **Having complicated dependencies between teams that slow them down:** Suboptimal organizational design and lack of awareness of all the links between systems and people can turn team coordination into a bureaucratic nightmare.
- **Creating fragile, unsustainable team structures:** Having only one or two developers in some critical technology can make your team as resilient as a chocolate teapot.

In summary, while architects stay close to the technology, they must ensure it's working for the business and customers, not the other way around. By keeping everyone in the loop and aligned, architects help steer clear of the many pitfalls of misalignment and keep the organizational machine running smoothly.

10.2: Superglue Abilities

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

- Skills
- Impact
- Leadership

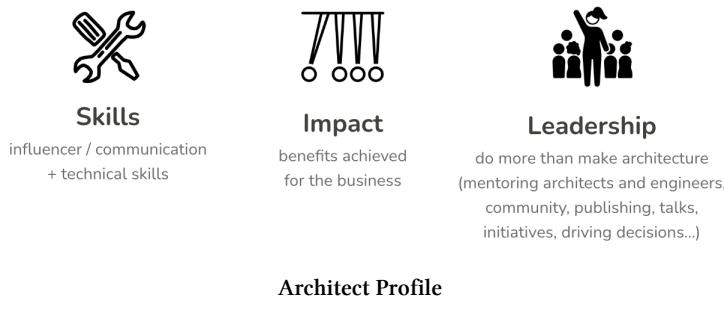


Figure 4: Architect Profile: Skills + Impact + Leadership.

10.2.1: Skills

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

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

10.2.2: 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.

10.2.3: Leadership

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

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

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

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

10.2.4: Balanced Development

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

In summary, developing architects as “superglue” means fostering a balanced combination of skills, impact, and leadership. Just like a three-legged stool, if one leg is too short, the whole thing topples over—quite possibly spilling your coffee in the process.

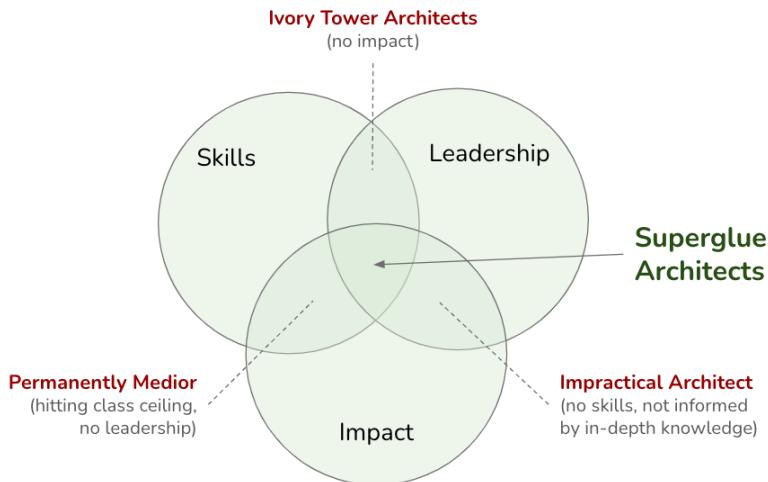


Figure 5: Architects must have a minimal “length” of all “legs” to be successful.

10.3: Questions to Consider

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

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

11: Skills

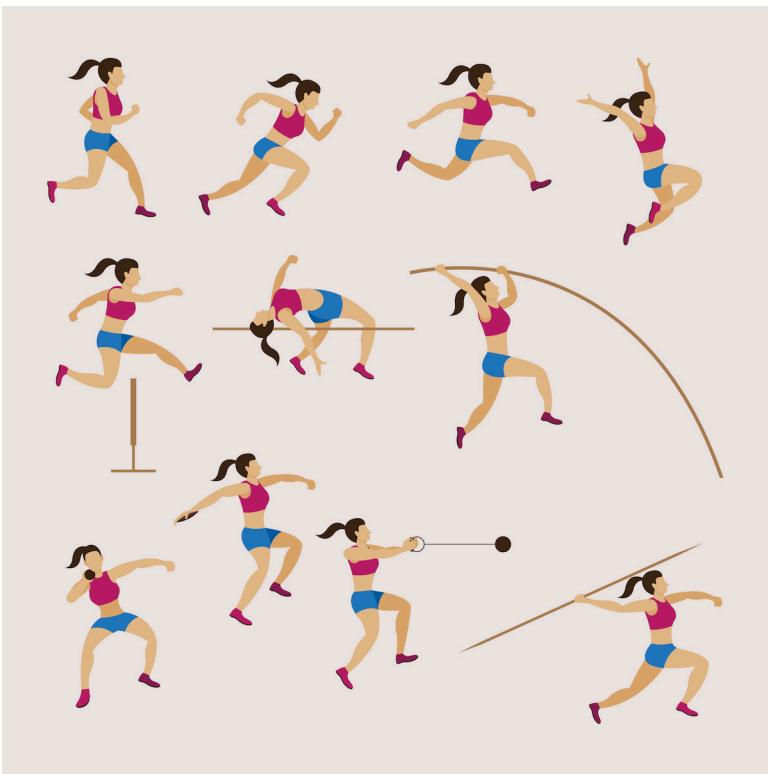


image by istock

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

KEY POINTS:

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

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

More specifically, a typical skillset of an architect includes:

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

11.1: Hard Skills

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

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

¹<https://obren.io/tools>

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

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

⁴https://obren.io/tools?tag=design_patterns

⁵https://obren.io/tools?tag=design_tactics

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

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

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

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

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

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

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

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

11.2: Soft Skills

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

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

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

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

¹³<https://obren.io/tools/sowhat/>

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

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

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

- **Organizational and time management skills¹⁶:** These skills involve the ability to efficiently plan, prioritize, and manage tasks, resources, and time. Effective organization and time management are crucial for meeting deadlines, achieving goals, and maintaining a healthy work-life balance. Key aspects of these skills include prioritization, goal-setting, task management, and delegation.
- **Analytical, strategic thinking, and problem-solving skills¹⁷:** Analytical skills involve assessing and interpreting complex information to make informed decisions. Strategic thinking is the capacity to envision and plan for long-term success, while problem-solving skills include identifying and addressing challenges creatively and effectively. These skills are essential for recognizing and capitalizing on unique opportunities and creating organizational value.

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

¹⁷<https://obren.io/tools?tag=it>

11.3: Product Development Skills

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

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

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

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

11.4: Business Skills

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

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

¹⁸<https://personalmba.com/>

¹⁹https://obren.io/tools?tag=domain_models

11.5: Decision-Making Skills

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

Architects will have two roles concerning decision-making:

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

Essential decision-making skills include:

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

11.6: Questions to Consider

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

12: Impact



image by 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**. Architects typically make an impact by:

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

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

12.1: Pillars of Impact

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

I use Staff Engineering roles as an inspiration for the development of architects. Tanya Reilly's book **The Staff Engineer's Path**¹ and Will Larson's book **Staff Engineer: Leadership beyond the management track**² are helpful guides in defining the responsibilities of architects.

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

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

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

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

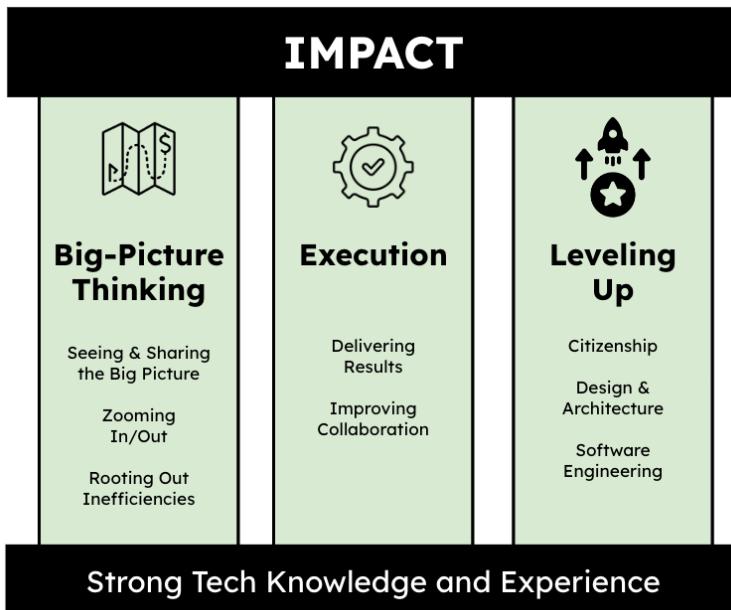


Figure 1: Key competencies of architects. Inspired by The Staff Engineer's Path by Tanya Reilly.

12.1.1: Big-Picture Thinking

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

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

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

12.1.2: Execution

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

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

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

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

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

12.1.3: Leveling Up

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

12.1.3.1: Citizenship

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

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

12.1.3.2: Design and Architecture

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

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

12.1.3.3: Software Engineering

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

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

12.2: Questions to Consider

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

13: Leadership



image by david mark from pixabay

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

KEY POINTS:

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

My approach to architecture leadership draws inspiration from two sources: **David Marquet's¹ leadership principles**, as articulated in his book “Turn the Ship Around!” and Netflix's valued behaviors. Marquet's ideas emphasize empowering team members, providing clarity, decentralizing decision-making, striving for continuous improvement, and practicing servant leadership. Meanwhile, **Netflix's valued behaviors²** offer useful guidance for coaching and developing architects aligned with the “super glue” version.

¹<https://davidmarquet.com/>

²<https://jobs.netflix.com/culture>

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

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

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

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

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



image by istock

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

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

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

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

13.2: Netflix's Valued Behaviors: Leadership Behaviors

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

13.2.1: Communication

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

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

13.2.2: Judgment

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

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

³<https://jobs.netflix.com/culture>

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

13.2.3: Impact

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

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

13.2.4: Inclusion

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

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

13.2.5: Selflessness

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

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

13.2.6: Courage

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

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

13.2.7: Integrity

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

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

13.2.8: Curiosity

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

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

13.2.9: Innovation

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

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

13.2.10: Passion

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

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

13.3: Questions to Consider

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

14: Architects' Career Paths: Raising the Bar

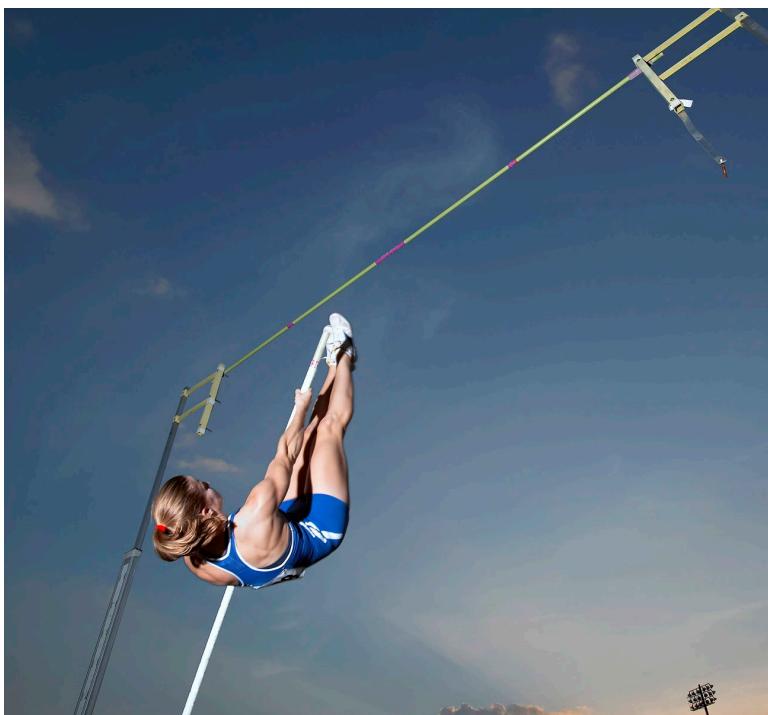


image by istock tableathny (cc by 2.0)

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

KEY POINTS:

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

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

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

14.1: Typical Architect's Career Paths

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

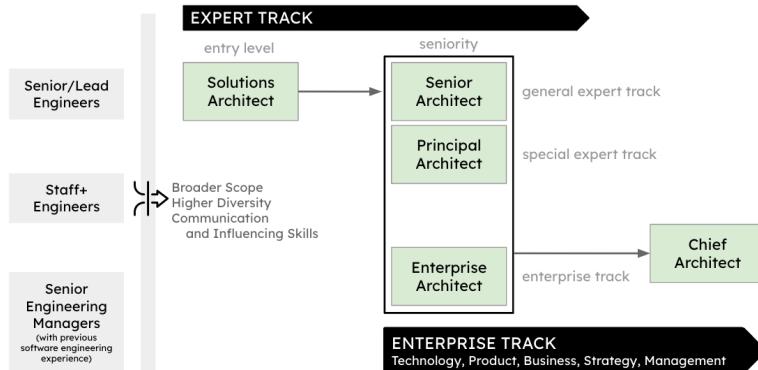


Figure 1: An example of an IT architect career paths in relation to engineering.

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

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

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

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

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

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

14.2: Hiring Architects

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

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

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

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

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

Step 1: Initial Screening Interview with Chief Architect

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

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

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

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

- Typical duration 90 min

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

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

Step 4: Non-technical stakeholders evaluation

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

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

14.3: Questions to Consider

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

Part III: Doing Architecture: Inspirations

15: Doing Architecture: Introduction



image by enrique meseguer from pixabay

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

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

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

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

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

16: The Culture Map: Architects' Culture Compass



image by maik from pixabay

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

KEY POINTS:

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

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

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

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

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

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

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

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

16.1: Communicating

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



image by istock

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

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

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

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

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



US Netherlands



Spain Italy



China



Australia Germany Denmark



Brazil France India



Canada



UK



Indonesia



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

16.2: Evaluating

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



image by rickey123 from pixabay

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

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

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

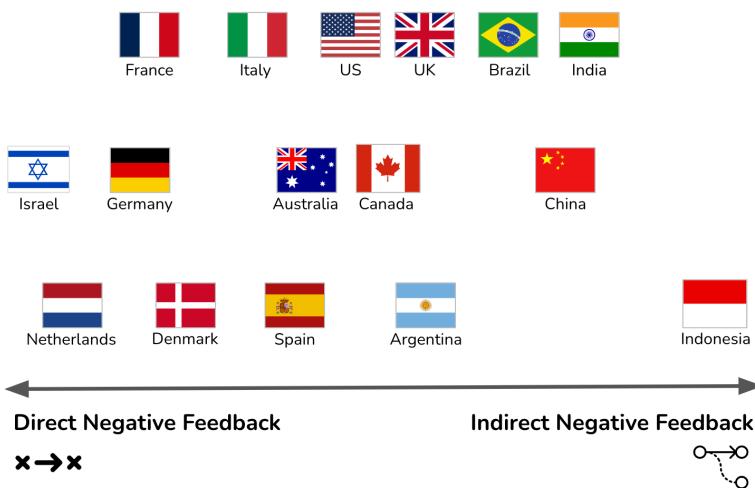


Figure 2: *The Evaluating scale: a graphical representation of some Culture Map countries (the cultural context of my recent professional interactions).*

16.3: Persuading

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



image by istock

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

Architects must be able to persuade in both applications-first

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

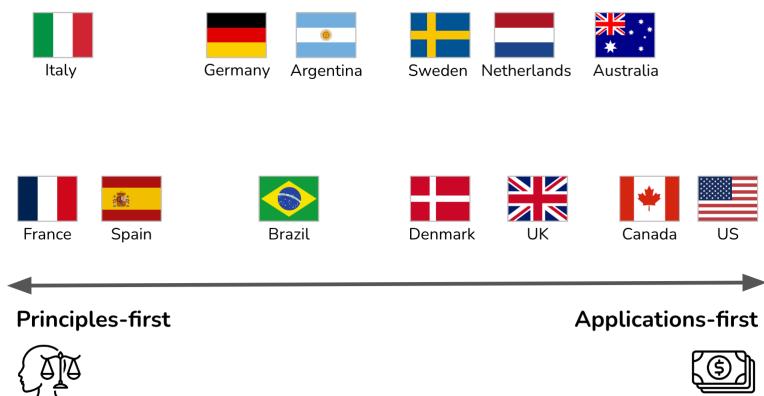


Figure 3: *The Persuading scale: a graphical representation of some Culture Map countries (the cultural context of my recent professional interactions).*

16.4: Leading

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

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



image by istock

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

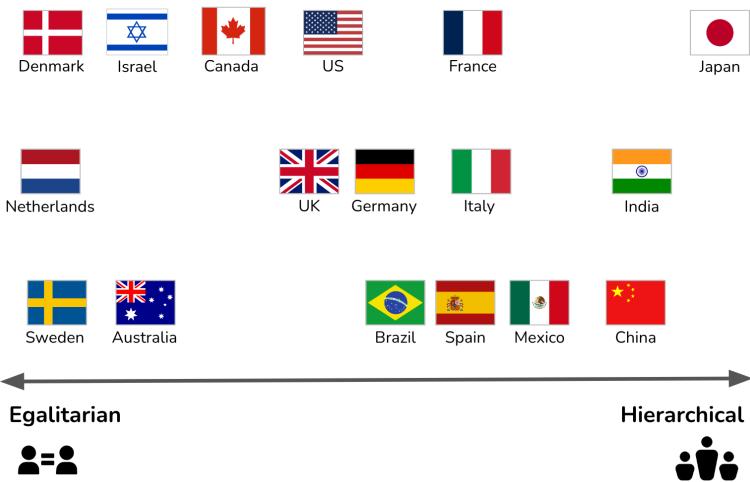


Figure 4: The Leading scale: a graphical representation of some Culture Map countries (the cultural context of my recent professional interactions).

16.5: Deciding

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



image by istock

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

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

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

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

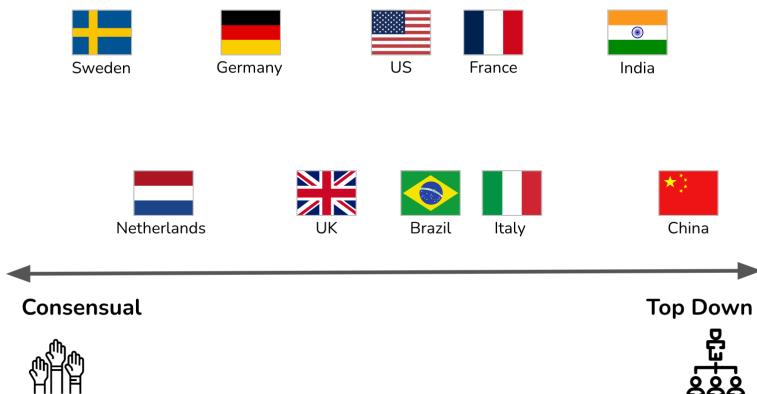


Figure 5: The Deciding scale: a graphical representation of some Culture Map countries (the cultural context of my recent professional interactions).

16.6: Trusting

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



image by istock

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

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

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

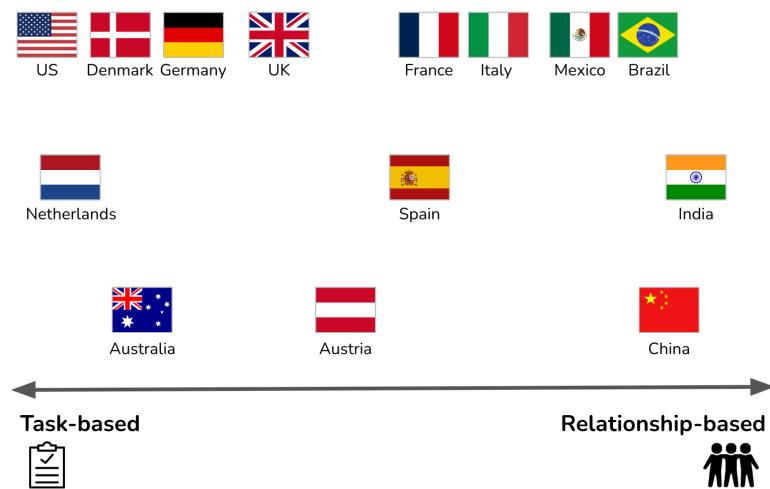


Figure 6: *The Trusting scale: a graphical representation of some Culture Map countries (the cultural context of my recent professional interactions).*

16.7: Disagreeing

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



image by istock

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

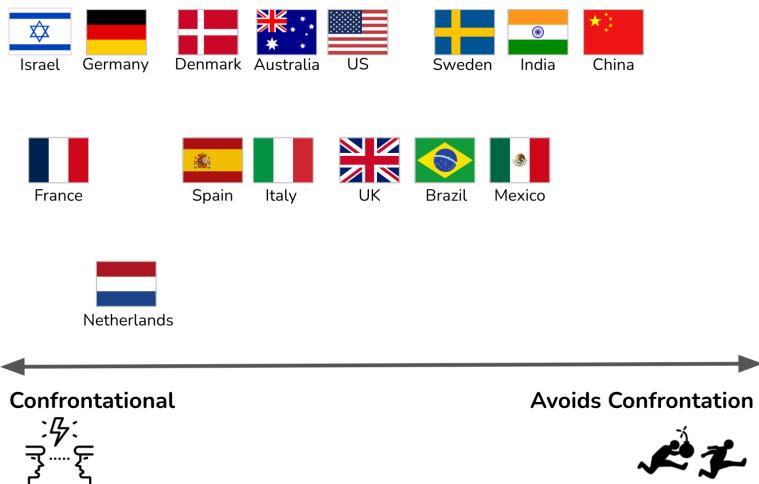


Figure 7: The Disagreeing scale: a graphical representation of some Culture Map countries (the cultural context of my recent professional interactions).

16.8: Scheduling

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



image by istock

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

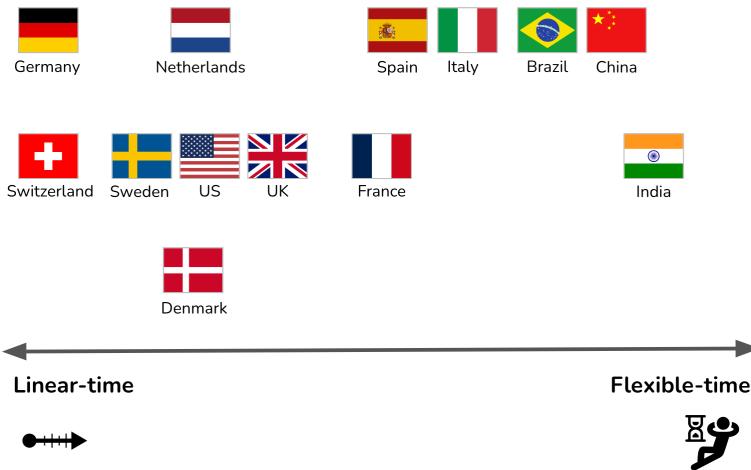


Figure 8: The Scheduling scale: a graphical representation of some Culture Map countries (the cultural context of my recent professional interactions).

16.9: Rules

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

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

16.10: Questions to Consider

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

17: Managing Organizational Complexity: Six Simple Rules



image by nat aggiate from pixabay

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

KEY POINTS:

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

The book *Six Simple Rules: How to Manage Complexity without Getting Complicated* by Yves Morieux and Peter Tollman offered fresh air for my vision of 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 Six Simple Rules approach is a practical solution for today's complex business environment. It 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**¹ 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

¹<https://martinfowler.com/bliki/ConwaysLaw.html>

are all about managing complexity without getting tangled up.

17.1: Background: Limitations of Hard and Soft Management Approaches

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

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

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

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

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

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

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

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

17.2: Six Simple Rules Overview

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

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

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

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

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

17.3: Rule 1: Understand What Your People Do

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



image by istock

17.3.1: 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—such as structure, scorecards, systems, and incentives—shape these goals, resources, and constraints.

17.3.2: Leveraging Architecture Practice

Architecture practice can significantly aid in understanding organizational behaviors through:

- Establishing a **Data Foundation** with an overview of various data sources to reveal where activities occur, visible trends, and cooperation among people. One principle of the Data Foundation, **build maps, not control units**, supports understanding and orientation rather than merely serving as a metric tool.
- Utilizing the **People Foundation** to connect individuals and enable them to learn about activities in different parts of the organization.

17.4: Rule 2: Reinforce Integrators

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



image by robert_owen_wahl from pixabay

17.4.1: General Guidelines

You can reinforce integrators by:

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

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

17.4.2: The Role of Architecture Practice

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

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

17.5: Rule 3: Increase the Total Quantity of Power

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



image by istock

17.5.1: General Guidelines

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

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

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

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

17.5.2: The Role of Architecture Practice

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

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

17.6: Rule 4: Increase Reciprocity

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



image by istock

17.6.1: General Guidelines

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

- **eliminating monopolies,**

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

17.6.2: The Role of Architecture Practice

Architecture practice can be directly related to increasing reciprocity:

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

17.7: Rule 5: Extend the Shadow of the Future

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



image by joe from pixabay

17.7.1: General Guidelines

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

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

17.7.2: The Role of Architecture Practice

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

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

17.8: Rule 6: Reward Those Who Cooperate

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



image by stocksnap from pixabay

17.8.1: General Guidelines

To reward those who cooperate, managers:

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

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

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

17.8.2: The Role of Architecture Practice

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

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

17.9: Questions to Consider

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

18: Effortless Architecture



image by istock

IN THIS SECTION, YOU WILL: Get summary of lessons learned from the Effortless book on how you functionally structure your work to make the most 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**"¹ 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**".² 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 im-

¹<https://gregmckeown.com/books/effortless/>

²https://en.wikipedia.org/wiki/No_Silver_Bullet

portance 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, 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.”*

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 Ford³](#), ~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.

Greg McKeown’s book summarizes many well-known practices into a pragmatic framework with a mindset of effortlessness. As such, 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

³<https://se-radio.net/2017/04/se-radio-episode-287-success-skills-for-architects-with-neil-ford/>

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 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.

18.1: 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.

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.

18.1.1: 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?*" 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. This simple yet powerful question can transform your approach, making tasks more manageable and less daunting. Or, as Kent Beck famously stated, for each desired change, **make the change easy, then make the easy change.**

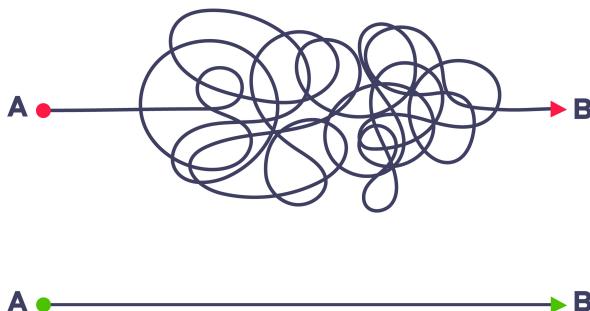


image by istock

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 inspiration to help their organizations make things easier:

- **Simplifying Code Maintenance With Clean Design:** Creating a modular source code design with clear, simple, and well-tested functions makes code maintenance easier. Clean and

modular code leverages existing libraries and prioritizes readability and maintainability over micro-optimizations. These widely known practices simplify maintenance, reducing the time and effort needed for debugging and updates. Still, developers are not universally following them.

- **Streamlining Deployment Processes:** Teams implement Continuous Integration/Continuous Deployment (CI/CD) pipelines using tools like Jenkins, GitHub Actions, or GitLab CI. Automating testing, building, and deployment processes minimizes human error and expedites the deployment process, making it more reliable and significantly less effortful.
- **Simplifying User Interface (UI) Design:** Designers and front-end developers prioritize user-centered design principles, emphasizing ease of use and intuitive navigation. They use UI frameworks like Material Design to create consistent and straightforward interfaces. A simplified and user-friendly UI enhances user experience and reduces the cognitive load on developers and end-users, making the application more straightforward to use and maintain.

By embracing the Effortless Inversion mindset, professionals can simplify their workflows, ultimately achieving better outcomes with less effort. This approach prioritizes simplicity, automation, and clarity.

18.1.2: 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 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 and Code Reviews as Collaborative Learning Sessions:** Architecture and code reviews, often seen

as tedious and time-consuming, can be transformed into collaborative learning sessions. Pair programming or review parties allow team members to discuss and review code over snacks and drinks. Adding a gamification element, such as rewarding the most insightful feedback, makes code reviews social and enjoyable. This engagement leads to better code quality 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.
- **Planning Meetings:** Project planning meetings, often long and dry, can be transformed into creative brainstorming sessions. Using mind maps, collaborative whiteboards, and interactive tools like Miro or Trello, along with fun ice-breakers and creative problem-solving exercises, keeps the team engaged and energized. This integration of creativity leads to more innovative solutions and a more enjoyable planning process.
- **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.

18.1.3: 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 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 certain technology stack or making architectural decisions that later proved suboptimal. Accept that every decision was 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.
- **Releasing Unrealistic Expectations:** Setting unrealistic expectations for project timelines or deliverables can lead to frustration and burnout. Set realistic, achievable goals and be flexible with adjustments as needed, understanding that unforeseen challenges are a natural part of the development process. When frustrated by unmet expectations, remind yourself of the progress made and express gratitude for the effort and achievements thus far. Letting go of unrealistic expectations reduces stress and enhances the ability to adapt and respond to challenges effectively.
- **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.

- **Letting Go of the Need to Control Every Detail:** Micro-managing every aspect of a project can lead to inefficiencies and a lack of autonomy for team members. Trust your team and delegate responsibilities effectively, focusing on guiding and supporting rather than controlling every detail. When you feel the urge to micromanage, remind yourself of your team's capabilities and express gratitude for their expertise and contributions. Letting go of the need for control empowers team members, fosters trust, and leads to a more efficient and autonomous work environment.

18.1.4: 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 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:

- **Avoiding Burnout Through Structured Work Sessions:** Software engineers often work long hours, leading to burnout and decreased productivity. Breaking down the workday into a limited number of focused sessions effectively combats this. Between these sessions, taking 15-30 minute breaks to rest and recharge helps maintain energy levels. For example, a morning session might be dedicated to coding, a mid-morning session for meetings or code reviews, and an afternoon session for testing or debugging. This structured

approach helps sustain high energy levels and concentration, reducing burnout risk and ensuring sustainable productivity.

- **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.

18.1.5: 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 and mindfulness.



image by 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 and mindfulness 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. Categorize your tasks into four quadrants: urgent and important, important but not urgent, urgent but not important, and neither. Prioritize tasks in the “important but not urgent” quadrant. Focusing on essential tasks enhances long-term productivity and progress, preventing the constant firefighting of urgent but less impactful tasks.
- **Being Fully Present in Team Interactions:** Engineers may be physically present but mentally preoccupied with other thoughts or tasks during team interactions. Cultivating mindfulness helps stay present in team interactions, making each person feel valued and heard. Before a meeting, take a few deep breaths to center yourself. During the interaction, consciously remind yourself to focus on the speaker and the discussion. Full presence in team interactions fosters deeper connections and understanding, leading to more effective collaboration and problem-solving.

18.2: 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.

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.

18.2.1: 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. 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 can align your efforts and provide a

clear direction. Or 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 forward momentum.



image by 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.

- **Completing a Sprint in Agile Development:** Agile sprints sometimes lack clear completion definitions, resulting in incomplete work and unfinished user stories. To mitigate this, establish a definition of “done” for each user story within the sprint, ensuring it meets the acceptance criteria, passes tests, and is ready for deployment. During sprint planning, define “done” for each user story: “User story Y is done when it meets the acceptance criteria, passes all automated tests, and is merged into the main branch.” This clarity ensures that each user story is fully completed by the end of the sprint, enhancing the overall quality and reliability of the product.
- **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 Documentation:** Documentation tasks can often seem never-ending, leading to procrastination and incomplete guides. To make this more manageable, specify what constitutes complete Documentation, such as covering all key features, including examples, and undergoing peer review. Set clear goals: “Documentation for Module A is done when it includes descriptions of all features, usage examples, and has been reviewed by a team member.” Clear definitions

prevent overextension and ensure that Documentation is comprehensive and valuable.

- **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.

18.2.2: 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.

- 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 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 consid-

ered 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 creates a starting point for adding new information.
- **Starting a Project Planning Session:** In planning the next sprint in an Agile development cycle, the sprint planning is complete when the sprint backlog is defined, tasks are estimated, and team members are assigned. The first obvious action is to schedule a sprint planning meeting. This action involves opening your calendar application and setting up a meeting invite for all team members. Scheduling the meeting sets the stage for collaborative planning.

18.2.3: 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 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.

- **Simplifying User Support:** Providing user support for a software product traditionally involves detailed support tickets, extensive troubleshooting steps, and multiple layers of escalation. The simplified approach starts with zero by eliminating unnecessary layers of support and streamlining user interactions. Implementation includes creating a comprehensive FAQ and knowledge base to address common issues and implementing chatbots for initial support queries to handle simple problems automatically. Reducing the complexity of user support processes increases efficiency, speeds up resolution times, and enhances user satisfaction.
- **Efficient Project Planning:** Planning a new software development project involves detailed plans, comprehensive timelines, and multiple approval stages. The simplified approach starts with zero, removing non-essential planning steps and focusing on core deliverables. The implementation uses a lean project management approach with simple Kanban boards to track tasks and progress, prioritizing tasks that directly contribute to the project goals. Simplifying project planning reduces overhead, keeps the team focused on delivering value and allows for greater flexibility and adaptability.
- **Streamlining Code Reviews:** Conducting code reviews for a new feature traditionally involves lengthy review processes with multiple reviewers and detailed scrutiny of every line of code. The simplified approach starts with zero by eliminating unnecessary layers of review. Implementation establishes clear coding standards and guidelines, uses automated code review tools to handle basic checks, and assigns one primary reviewer for each pull request. Streamlined code reviews increase efficiency, reduce bottlenecks, and ensure consistent

code quality without excessive overhead.

18.2.4: 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 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 edge cases and optimizations for now. This "rubbish" version allows you to quickly identify the main challenges and requirements, setting a foundation for iterative improvements.

- **Writing Initial Documentation:** When documenting a new software module, understand that the first draft will not be perfect. Begin with zero-draft writing, jotting down rough notes and ideas without worrying about structure or grammar. Open a document and write down everything users need to know in any order. This raw documentation provides a base that can be refined and organized later, ensuring that all important information is captured early on.
- **Designing a User Interface:** When creating a new user interface (UI) for an application, acknowledge that the initial design might look unpolished and lack visual appeal. Start with zero-draft design, sketching out basic wireframes on paper or using simple digital tools, focusing on layout and functionality rather than aesthetics. Use tools like Balsamiq or pen and paper to create quick, rough UI sketches. These preliminary designs allow for rapid feedback and iteration, helping to refine the UI based on functionality and user experience.
- **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.
- **Early Testing Phases:** When testing a new application or feature, recognize that initial tests might not cover all scenarios and could miss critical bugs. Start with zero-draft testing, conducting basic tests to identify apparent issues and verify that the main functionality works. Write simple test cases and execute them manually or with basic automated scripts. These early tests help catch glaring problems and provide a

baseline for developing more comprehensive test cases later.

- **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.
- **Prototyping a New Algorithm:** When developing a new algorithm for data processing, understand that the first version may be inefficient and error-prone. Start with a zero-draft algorithm, writing a basic version to solve the problem in the simplest way possible. Implement the algorithm with straightforward logic, focusing on functionality rather than performance. This prototype allows for early testing and validation of the algorithm's approach, which can be optimized and refined through subsequent iterations. Or, as wisely noted by Donald Knuth, "*premature optimization is the root of all evil.*"

18.2.5: 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 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.

- **Writing Technical Documentation:** When documenting a software module, the traditional approach might involve writing extensive documentation in a single session, resulting in low-quality output. A balanced approach involves writing no less than 500 words and no more than 1500 words per day. Break the documentation into sections and tackle each in a focused session, taking breaks as needed. This approach ensures consistent, high-quality documentation that is clear and comprehensive.
- **Conducting Code Reviews:** When reviewing a large pull request from a teammate, the traditional approach might involve attempting to review all changes in one sitting, leading to oversight and fatigue. A balanced approach commits to examining a range of files per session. Break the review into manageable chunks, and take breaks to reflect and refresh. This deliberate pace allows for more thorough and thoughtful reviews, improving code quality and collaboration.
- **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.
- **Sprint Planning in Agile Development:** The traditional approach to planning the tasks for the next sprint often involves rushing through planning meetings to get back to coding quickly. A balanced approach allocates no less than 1 hour and no more than 2 hours for sprint planning. Schedule focused sprint planning sessions with clear agendas and objectives. This balanced pace ensures comprehensive

planning, resulting in realistic and achievable sprint goals.

- **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.

18.3: 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.

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.

18.3.1: 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 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:

- **Understanding First Principles in Algorithm Design:** In optimizing a search algorithm, the approach begins with first principles. Learn the fundamental principles of algorithms, including time complexity, space complexity, and basic search techniques like binary search and hash tables. Leverage the works of pioneers like Donald Knuth and their analysis of algorithms. You can choose or design an efficient search algorithm by understanding why certain algorithms perform better under specific conditions. Grounding yourself in first principles allows for creating or adapting algorithms optimized for your particular use case, achieving better performance with less effort.
- **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 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.
- **Enhancing Cybersecurity Measures:** When enhancing an organization's cybersecurity measures, start with the first principles by understanding the core concepts of cybersecurity, including encryption, network security, access controls, and threat detection. Follow guidelines and frameworks established by leading cybersecurity organizations. Implement security measures such as multi-factor authentication, regular security audits, and the latest encryption standards. Building on fundamental principles and expert knowledge enables the efficient implementation of robust security measures, reducing the risk of breaches.

18.3.2: 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 not only reinforces your own understanding but also 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 underscores the crucial role you play in knowledge sharing and the impact you can have on others.

One motivation for writing this book is to create reusable material that teaches others about pragmatic approaches to running architecture practices and give them material they can reuse to teach others the same.



image by 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 inspiration to help their organizations harness the strength of ten:

- **Teaching Code Reviews:** A senior developer wants to improve the code review process within the team. Instead of just conducting code reviews, the senior developer holds a workshop to teach team members how to conduct effective code reviews themselves. Participants are encouraged to practice reviewing code and receiving feedback on their technique using real-world examples and stories from past projects to illustrate key points. The outcome is that team members become proficient in conducting code reviews, multiplying the senior developer's impact, and improving overall code quality.
- **Sharing Debugging Techniques:** An engineer skilled at debugging complex issues decides to conduct training sessions to share debugging strategies with the team. Using case studies from actual bugs to demonstrate techniques, the engineer creates a guide outlining common debugging steps and tools. Team members are encouraged to present their own debugging cases and discuss approaches. By teaching others how to debug effectively, the engineer empowers the team to solve problems more independently, leading to faster resolutions and less downtime.
- **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 API docs, 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.
- **Cultivating a Culture of Testing:** An engineer with strong skills in automated testing wants to instill a testing culture within the team. The engineer organizes training sessions focused on the principles and practices of automated testing using tools like Selenium, JUnit, or Pytest. The engineer then demonstrates how to write and maintain automated tests and shares personal stories about how automated testing has prevented bugs and improved software quality. As a result, team members learn to value and implement automated testing, leading to higher software quality and fewer production issues.

18.3.3: 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 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. The approach involves high-tech automation by integrating 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.

- **Using Checklists for Routine Tasks:** Repetitive tasks like setting up development environments or conducting code reviews involve many steps that can be easily forgotten. The approach involves low-tech automation by creating detailed checklists for routine tasks. Use tools like Google Keep, Evernote, or simple printed checklists to ensure all steps are followed. The outcome is that checklists ensure consistency and accuracy, reducing the mental load required to remember every step.
- **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 configurations, 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.

- **Automating Email Management:** Managing and sorting through emails can be a significant time drain. The approach involves high-tech automation using email filters and automation tools like Gmail filters or Outlook rules. Set up filters to automatically sort incoming emails into folders, label important messages, and archive or delete unwanted emails. The outcome is automated email management that reduces inbox clutter and helps you focus on important communications.

18.3.4: 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 architecture practice are used or misused to force bureaucratic controls as teams often do not trust each other.



image by 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:

- **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.
- **High-Trust Agreements in Project Management:** Starting a new project with a distributed team requires establishing a

high-trust agreement. Set clear agreements on results, roles, rules, resources, and rewards at the project's outset. Hold a kickoff meeting to outline project goals, define individual responsibilities, set standards for communication and deliverables, allocate resources, and agree on how achievements will be recognized. The outcome is clear expectations and defined roles that reduce misunderstandings and ensure everyone is aligned, fostering a collaborative and productive work 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, trust their capability to handle the task while providing guidance and support as needed. 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.
- **Building a High-Trust Structure:** To resolve conflicts between development and operations teams, implement DevOps practices to align goals and improve collaboration. Define shared goals and metrics for success, such as deployment frequency and system uptime. Use automation tools to streamline processes and reduce friction points. Hold

joint meetings to discuss priorities and challenges. The outcome is a high-trust structure that promotes collaboration, reduces conflict, and increases the efficiency of deploying and maintaining software.

18.3.5: 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.

Don't just manage problems—solve them before they happen. Seek simple actions today that can prevent complications tomorrow. By investing a small amount of effort now, you 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 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:

- **Implementing Automated Testing:** Repeatedly encountering bugs in production that were not caught during manual testing calls for preventative action by implementing automated testing. Set up a suite of automated tests using frameworks like Selenium for UI testing, JUnit for unit tests, and pytest for Python applications. Integrate these tests into the CI/CD pipeline. The outcome is that automated tests catch issues early in the development cycle, reducing the likelihood of bugs reaching production and saving time on manual testing.

- **Regular Code Reviews:** Inconsistent code quality and frequent issues from unreviewed code changes necessitate preventative action by establishing a regular code review process. 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:** Frequently discovering system outages or performance issues after they impact users requires preventative action by implementing monitoring and alerting systems. 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.
- **Creating Detailed Documentation:** New team members struggle to understand the system architecture and workflows, leading to onboarding delays and calls for preventative action by creating and maintaining detailed documentation. Document system architecture, workflows, coding standards, and common troubleshooting steps. Use tools like Confluence or GitHub Wikis to keep documentation accessible and up-to-date. The outcome is comprehensive documentation that speeds up onboarding, reduces the learning curve for new team members, and serves as a reference for existing members, preventing repetitive questions and confusion.
- **Regular Security Audits:** Discovering security vulnerabilities after a breach, leading to significant downtime and data loss, requires preventative action by 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 con-

duct periodic security reviews. The outcome is that regular security audits and training reduce the risk of vulnerabilities, preventing potential breaches and ensuring data protection.

- **Using Configuration Management:** Inconsistent configurations across different environments cause deployment failures and bugs, which necessitate preventative action by implementing configuration management. Use tools like Ansible, Chef, or Puppet to manage and automate environment configurations. Store configuration files in version control. The outcome is consistent configurations across environments, reducing deployment issues and ensuring that applications run smoothly in different environments.
- **Scheduling Regular Backups:** Data loss due to hardware failures or accidental deletions requires preventative action, such as implementing a regular backup schedule. Use backup solutions like AWS Backup, Veeam, or custom scripts to back up critical data regularly. Test backup restoration periodically. The outcome is regular backups, ensuring data can be quickly restored in case of loss, minimizing downtime, and preventing significant data loss.
- **Implementing Continuous Integration:** Integration issues arising from merging code changes at the end of the development cycle necessitate preventative action by implementing Continuous Integration (CI). Use CI tools like Jenkins, Travis CI, or CircleCI to integrate and test changes as they are made automatically. Ensure the CI pipeline includes build, test, and deployment stages. The outcome is continuous integration, catching integration issues early, ensuring that code changes work well together, and reducing the complexity of late-stage integrations.

18.4: 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. By adopting a mindset that prioritizes ease and enjoyment, you enable a more streamlined and fulfilling approach to achieving your goals.

18.5: 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?

19: Effective Communication



image by istock

IN THIS SECTION, YOU WILL: Get a summary of several resources to help you communicate more effectively, provide good feedback, and lead tough conversations..

KEY POINTS:

- “So What! How to Communicate What Really Matters to Your Audience” by Mark Magnacca focuses on tailored communication for IT professionals. It emphasizes the importance of relevance and audience understanding to enhance the effectiveness of technical discussions.
- “Radical Candor” by Kim Scott provides a framework for IT leaders to combine personal empathy with direct challenges to foster robust team dynamics and honest communication, which is crucial for project success and team development.
- “Never Split the Difference” by Chris Voss introduces negotiation techniques from high-stakes FBI scenarios, adapted for IT and software architecture discussions, to help professionals achieve better outcomes through strategic empathy and questioning.

This text presents three practical and influential resources I've found beneficial for IT and software architects to enhance their communication skills.

- “So What! How to Communicate What Really Matters to Your Audience” by Mark Magnacca focuses on communication techniques emphasizing clarity, audience understanding, and relevance. It teaches how to convey ideas effectively by understanding the audience’s needs, utilizing clear messaging, and engaging through stories and interactive dialogue. The book provides specific storylines and scenarios to apply these techniques.
- “Radical Candor: Be a Kick-Ass Boss Without Losing Your Humanity” by Kim Scott - This guide is essential for anyone who aims to build strong teams and foster open

communication. Scott introduces the concept of Radical Candor, which combines personal care with direct challenges, helping leaders provide honest feedback while building genuine relationships. The book outlines various quadrants of interaction styles, including Ruinous Empathy and Obnoxious Aggression, providing a framework for understanding the effects of different communication approaches.

- “Never Split the Difference: Negotiating As If Your Life Depended On It” by Chris Voss - Drawing from his experience as an FBI negotiator, Voss offers strategies for high-stakes negotiations. Techniques such as Tactical Empathy, Mirroring, and Calibrated Questions can help navigate difficult discussions and reach effective agreements without compromising goals.

Each resource can equip IT architects with the tools to improve their communication and leadership skills, which are crucial for influencing stakeholders, driving successful project outcomes, and managing teams in the fast-paced technology field.

19.1: So What!

“So What? How to Communicate What Really Matters to Your Audience” by Mark Magnacca is an essential guide for effectively communicating their ideas and solutions to stakeholders, team members, or clients. This book emphasizes the importance of delivering clear, concise, and relevant messages that resonate with the audience’s needs and concerns. By applying the principles outlined in “So What!”, IT architects can enhance their communication skills, ensuring their technical insights are understood and valued.



image by istock

19.1.1: Key Concepts from the Book

The book underscores the paramount importance of **knowing your audience**. This understanding, which includes their interests, concerns, and motivations, is the bedrock of effective communication. It also emphasizes the necessity of focusing on the “**So What?**” factor, urging communicators always to clarify their messages’

relevance and why the audience should care. **Clarity and brevity** are also essential—you should deliver messages straightforwardly, avoiding jargon and complex terms.

Another critical point is keeping the **audience engaged**; interactive communication helps maintain interest and promotes better understanding. Finally, **preparation and practice** are essential; rehearsing your communication ensures clarity and builds confidence in your delivery.

The **use of stories and analogies** is recommended to make technical concepts more relatable and easier to understand. Mark Magnacca also provides practical tips on communicating different types of messages. He describes **various kinds of storylines**¹ you can use to communicate almost any topic:

- **Action Jackson:** A simple storyline describing steps. Use it when you want to spell out an action plan. Build it to map overall recommendations and supporting steps. Avoid it when the audience still needs to be convinced.
- **The Pitch:** Articulates and supports a recommendation. Use it when you need to persuade someone. Highlight your value proposition ‘right up front.’ Avoid being glib; your supporting reasons must convince your audience.
- **Traffic Light:** Use it for status updates. It enables you to provide an overview and explain your current position in detail. Avoid being sloppy.
- **Close the Gap:** Use it to gain your audience’s buy-in in one meeting. Explain where you are and where you need to get. Build it to outline the case for your action plan. Stick to storyline rules so your audience supports your action plan.
- **Houston, We Have a Problem:** Use when you need to convince an audience of a problem and talk them through actions. Great at combining diagnostics and action. Maps

¹<https://obren.io/tools/sowhat/>

problems, the cause, and resulting steps. Avoid overkill; it is about convincing.

- **To B, or Not to B:** Use to take your audience on the journey through your thinking. It argues for one particular option. It enables you to explain first and then make recommendations last. Make sure it's complete - great when you need to work through all options, not just one option you like.
- **Watch Out:** This storyline persuades the audience of the need to change direction. It combines what's working, risks, and action in the same story. Maps what's succeeded, the risks, and remedies. Avoid crafting a narrative that flows without compelling logic.

The book also proposes a structured framework to communicate ideas clearly and persuasively. The framework suggests that each communication has three main parts:

- A short introductory narrative with three elements:
 1. **Context:** This sets the stage by providing the background information or the current situation. It helps the audience understand the environment or circumstances in which the issue or opportunity exists. Not everyone knows the situation as you do.
 2. **Trigger:** This specific event or observation prompts the need for discussion or action right now. It highlights what has changed or what new information has come to light that necessitates addressing the issue.
 3. **Question:** This is the pivotal "so what?" question that captures the main concern or the central issue that needs to be addressed.
- **One key point** that is a focus of your storyline. It should provide a clear and concise answer to the question from the introductory narrative.

- **Supporting points:** These support the proposed answer with data, examples, or logical reasoning. They provide the justification for the recommendation and help convince the audience of its validity.

This structure helps ensure that communication is clear, logical, and persuasive, making it easier for the audience to understand and buy into the proposed solutions.

19.1.2: Examples

Here are a few examples of how you could use “So What!” techniques in IT practice:

19.1.2.1: Example 1: Action Jackson

Scenario: Implementing a new feature in a software application.

Introductory Narrative:

- Context: Our previous analysis has shown that we must enhance our user experience to stay competitive.
- Trigger: Feedback from our recent user survey indicates a high demand for a new feature, an opportunity to improve the overall user experience and add value via new functionality.
- Question: How do we implement this feature effectively? Can we accelerate the delivery?

Key Point:

- We will prioritize this as a new initiative but not create shortcuts. We will follow our standard structured approach involving requirement gathering, design, development, testing, deployment, and post-deployment monitoring.

Supporting Points:

- As this is a complex and risky feature, we must follow a disciplined process.
- We will gather requirements by collaborating with stakeholders to define the feature, create design documents approved by the architecture team, implement the feature using our standard methodology, conduct thorough unit and integration testing, deploy the feature to production, and monitor its performance post-deployment.
- This process ensures a successful feature launch with minimal risks.

19.1.2.2: Example 2: The Pitch

Scenario: Proposing a new tool for automated testing.

Introductory Narrative:

- **Context:** Our current testing process is time-consuming and prone to human error. We have previously evaluated several tools and identified Tool X as the best option for automation. At that point, we decided not to adopt it to lack of resources.
- **Trigger:** Recent delays and errors in the last project highlighted the inefficiency of our manual testing, reopening requests to adopt an automation tool. However, adoption of the tool would require additional resources.
- **Question:** Why should we invest resources and adopt Tool X for automated testing?

Key Point:

- Tool X is a cost-effective solution aligned with our technology platform that will increase efficiency, save costs, enhance accuracy, and scale with future growth.

Supporting Points:

- Tool X reduces testing time by 50%, saves \$100,000 annually, minimizes human error for more reliable results, and can quickly scale to accommodate future growth.
- These benefits make Tool X a valuable addition to our testing toolkit.

19.1.2.3: Example 3: Traffic Light

Scenario: Providing a status update on a software development project.

Introductory Narrative:

- **Context:** Our project is midway through the development phase.
- **Trigger:** Stakeholders need an update on the project status.
- **Question:** What is the project's current status, and what actions are needed?

Key Point:

- The project is progressing overall, but the frontend development and integration testing areas require attention to stay on track.

Supporting Points:

- Backend development is on schedule and 80% complete (green).
- Frontend development is slightly behind due to resource constraints (yellow).
- Integration testing is delayed by unexpected bugs (red).
- To address these issues, we must resolve integration bugs and allocate additional resources to front-end development.

19.1.2.4: Example 4: Close the Gap

Scenario: Convincing stakeholders to approve a migration to a new cloud platform.

Introductory Narrative:

- **Context:** Our current infrastructure is costly and lacks scalability.
- **Trigger:** We must improve cost efficiency and scalability to close the gap with our competitors.
- **Question:** Why should we migrate to Cloud Platform Y?

Key Point:

- Migrating to Cloud Platform Y will improve cost efficiency and improve scalability, enabling us to onboard more new customers.

Supporting Points:

- Our current state involves high costs and limited scalability.
- By migrating to Cloud Platform Y, we will achieve significant cost savings and scalability improvements, ensuring we can meet future demands efficiently.

19.1.2.5: Example 5: Houston, We Have a Problem

Scenario: Addressing a critical performance issue in the software.

Introductory Narrative:

- **Context:** We use a legacy database with known performance issues. Until recently, database performance was manageable and did not negatively affect user experiences.

- **Trigger:** Since this week, users have reported slow response times, impacting their experience.
- **Question:** How do we resolve this performance issue?

Key Point:

- The performance issue is serious and will not disappear by itself. We need to take proactive steps to resolve it. We must thoroughly analyze, optimize queries, and reindex the database.

Supporting Points:

- The performance issue is due to inefficient database query optimization.
- We will diagnose the problem, pinpoint inefficient queries and indexing issues, implement optimizations, and continuously monitor performance to ensure improvement.

19.1.2.6: Example 6: To B, or Not to B

Scenario: Deciding between two software development frameworks.

Introductory Narrative:

- **Context:** We must select a framework for our next project.
- **Trigger:** The deadline for project start is next month.
- **Question:** Which framework should we choose?

Key Point:

- After evaluating both, Framework A is the best choice.

Supporting Points:

- Framework A offers high performance and extensive community support but has a steep learning curve and higher cost. Framework B is easier to use and cheaper but has limited scalability and a smaller community.
- Based on our requirements and long-term goals, Framework A aligns better despite its higher cost.

19.1.2.7: Example 7: Watch Out

Scenario: Advising on potential risks in a software project.

Introductory Narrative:

- **Context:** Our project is progressing, but with some minor manageable security risks.
- **Trigger:** The upcoming API integration opens significant new security risks.
- **Question:** How do we mitigate these risks?

Key Point:

- We need to implement additional security measures and conduct thorough testing.

Supporting Points:

- Although our development process is efficient and on schedule, the API integration introduces security risks.
- With our team's expertise, we can implement additional security measures and conduct thorough testing before integration, proactively addressing these risks and ensuring project success.

19.2: Radical Candor

“Radical Candor: Be a Kick-Ass Boss Without Losing Your Humanity” by Kim Scott is an invaluable resource for IT and software architects who engage in critical conversations. The book provides a framework for giving and receiving feedback, building solid relationships, and fostering a culture of open communication. These principles can greatly enhance team dynamics, project efficiency, and overall job satisfaction.



image by istock

19.2.1: Key Concepts from the Book

Scott defines a feedback-providing communication framework as having two access points: the directness of challenge and the amount of personal caring. Based on this framework, she describes four quadrants, representing different styles of how people provide feedback:

- **Radical Candor:** Caring personally while challenging directly.
- **Ruinous Empathy:** Caring personally without challenging directly.
- **Obnoxious Aggression:** Challenging directly without caring personally.
- **Manipulative Insincerity:** Neither caring personally nor challenging directly.

Radical Candor is the only favorable option, which only happens when you:

1. **Care Personally:** Show genuine concern for your team members' well-being and career growth.
2. **Challenge Directly:** Provide honest, straightforward feedback to help team members improve.

"Radical Candor" offers a framework that IT architects can use to enhance their leadership and communication skills. By applying Scott's principles of caring personally and challenging directly, IT professionals can build stronger teams, foster a culture of continuous improvement, and drive successful project outcomes. The contrasting examples from other quadrants (Ruinous Empathy, Obnoxious Aggression, and Manipulative Insincerity) illustrate the importance of balancing care and directness to achieve the best results.

19.2.2: Examples

Here are a few examples you could use to get inspiration on how to use Radical Candor in IT practice:

19.2.2.1: Scenario 1: Providing feedback on a poorly designed system architecture.

With Radical Candor, begin by acknowledging the hard work the team has put into the current design. “I appreciate the effort everyone has invested in this project. However, I’ve noticed some scalability concerns that we need to address. Let’s look at how we can refactor the design to handle increased load better.” In contrast, Ruinous Empathy might sound like, “Great job on the design, everyone. Let’s not worry too much about scalability issues right now; we’ll figure it out later.” Obnoxious Aggression would be, “This design is terrible. How could you miss these scalability issues? We need to fix this immediately.” Manipulative Insincerity would come across as “I guess the design is okay, but there might be some minor scalability issues. Do whatever you think is best.”

19.2.2.2: Scenario 2: Addressing technical debt and its impact on future projects.

Using Radical Candor, you could say, “I know everyone has been working tirelessly to meet our deadlines. We need to allocate time to reduce our technical debt. It will hinder our future development and slow us down if we don’t address it now.” On the other hand, Ruinous Empathy might be, “You all have worked so hard; let’s not worry about technical debt right now. We can deal with it later.” Obnoxious Aggression would sound like, “The technical debt is getting out of control. This is unacceptable, and we need to fix it now.” With Manipulative Insincerity, you might say, “There might be some technical debt, but it’s not a big deal. Just keep doing what you’re doing.”

19.2.2.3: Scenario 3: Providing feedback to a team member who consistently writes inefficient code.

Radical Candor might involve saying, “I’ve noticed you’re very dedicated to your work and always eager to help. However, I’ve seen some inefficiencies in your code that we need to improve. Let’s work together to optimize it and follow best practices.” Ruinous Empathy would be, “You’re doing a great job, keep it up! We can look at the code optimization some other time.” Obnoxious Aggression could sound like, “Your code is inefficient and needs to be fixed immediately. This is not acceptable.” Manipulative Insincerity might be, “Your code is fine, but there might be a few inefficiencies. It’s not a big deal, though.”

19.2.2.4: Scenario 4: Discussing the adoption of new coding standards with the team.

With Radical Candor, you could say, “You all have done an excellent job adapting to new standards and technologies in the past. To maintain code quality and ensure consistency, we need to adopt these new coding standards. I understand it’s an additional effort, but it will benefit us in the long run.” Ruinous Empathy might be, “You’re all doing great; there’s no need to change our coding standards right now.” Obnoxious Aggression would be, “We need to adopt these new coding standards immediately. I don’t care how much extra work it is.” Manipulative Insincerity would come across as, “These new coding standards are optional. Use them if you want, but it’s not a big deal if you don’t.”

19.2.2.5: Scenario 5: Addressing a conflict between team members regarding code ownership.

Using Radical Candor, you might say, “I appreciate the hard work both of you have put into this project. There seems to be some tension regarding code ownership. Let’s discuss how we can

collaborate more effectively and ensure everyone's contributions are recognized." Ruinous Empathy would be, "You're both doing great work; let's not worry about who owns what code. It's not important." Obnoxious Aggression could be, "This conflict over code ownership is ridiculous. Figure it out or I'll reassign the tasks." Manipulative Insincerity might be, "I'm not sure whose code is whose, but just try to get along and make it work."

19.2.2.6: Scenario 6: Encouraging a junior architect to take on more challenging tasks.

With Radical Candor, you could say, "I've seen a lot of potential in your work, and you've grown significantly since joining the team. I'd like to see you take on some more challenging tasks. It might be tough at first, but I believe it will help you develop your skills further." Ruinous Empathy might sound like, "You're doing well; no need to take on more challenging tasks right now. Just keep doing what you're comfortable with." Obnoxious Aggression would be, "You need to take on more challenging tasks. Stop taking the easy way out." Manipulative Insincerity would be, "You could try some more challenging tasks if you want, but it's fine if you don't feel up to it."

19.2.2.7: Scenario 7: Managing unrealistic project expectations from upper management.

Using Radical Candor, you might say, "I know there's a lot of pressure to deliver quickly in a competitive market. However, the current expectations are unrealistic given our resources. We must discuss what can be realistically achieved within the given timeframe and how we can adjust our approach." Ruinous Empathy would be, "We'll try our best to meet the deadlines, even if it's unrealistic." Obnoxious Aggression could be, "These expectations are completely unrealistic and unachievable. We can't do it." Manipulative Insincerity might be, "Sure, we'll do our best to meet

the deadlines, but it might be tough.”

19.2.2.8: Scenario 8: Fostering a culture of open feedback within the team.

With Radical Candor, you could say, “I want everyone to know that your contributions are highly valued and your well-being is important to me. To grow as a team, we need to be open and honest with each other. Please feel free to share your thoughts and feedback, whether positive or critical, so we can continuously improve.” Ruinous Empathy might be, “Everyone’s doing great, so there’s no need for much feedback. Let’s keep things as they are.” Obnoxious Aggression would be, “We need to be brutally honest with each other. If you can’t handle criticism, this isn’t your place.” Manipulative Insincerity might be, “Feel free to share your feedback if you want, but it’s not a big deal if you don’t.”

19.3: Never Split the Difference

“Never Split the Difference: Negotiating As If Your Life Depended On It” by Chris Voss is an essential read for IT and software architects who frequently face challenging conversations and negotiations—drawing from his experience as an FBI hostage negotiator, Voss shares effective techniques and strategies that you can apply directly to IT architecture and software architecture discussions, ensuring better outcomes without compromising on critical objectives.

“Never Split the Difference” means not compromising or settling for less than what you want in a negotiation. Instead of agreeing to a middle-ground solution where both parties give up something, Voss advocates using strategic and psychological techniques to achieve a more favorable outcome. The idea is to aim for the best possible deal rather than a mediocre one that leaves both sides partially dissatisfied. For instance, the teams can agree to work on two projects to avoid blame and tense discussion even if they know they lack capacity and the planning is too optimistic. They then fail both projects. Alternatively, they should reject or postpone one project and commit to delivering only one with a lower risk of failure.



image by istock

19.3.1: Key Concepts from the Book

The books apply several key concepts and techniques you can use in difficult negotiations:

1. **Tactical Empathy:** Understanding and actively listening to the counterpart's perspective to build rapport and trust.
2. **Mirroring:** Repeating the last few words the other person said to encourage them to elaborate.
3. **Labeling:** Identifying and acknowledging the other person's feelings to validate their emotions.
4. **Accusation Audit:** Preemptively addressing any negative assumptions the other party might have.
5. **Calibrated Questions:** Asking open-ended questions that begin with "What" or "How" to steer conversations and gather information.

6. **The Power of No:** Understanding that hearing “No” can be a pathway to uncovering the real issues and fostering a sense of control for the other party.

“Never Split the Difference” offers valuable techniques for IT and software architects to enhance their negotiation skills, enabling them to navigate difficult conversations and achieve favorable outcomes. By applying Voss’s strategies, IT professionals can address concerns, build consensus, and drive successful implementation of critical architectural changes.

19.3.2: Examples

Here are a few examples you could use to get inspiration on how to use described techniques in IT practice:

19.3.2.1: Scenario 1: Convincing stakeholders to adopt a new cloud architecture.

Use Tactical Empathy by understanding the concerns of stakeholders who may worry about data security or cost implications. You might say, “It sounds like you are concerned about the security of our data in the cloud.” Employ Mirroring when a stakeholder says, “I’m worried about the cost,” by responding, “The cost?” Utilize Labeling by saying, “It seems like you’re feeling uncertain about the transition.” Conduct an Accusation Audit with, “You might think this is going to be too expensive and risky, and I understand why you’d feel that way.” Ask Calibrated Questions such as, “What are the main challenges you see with moving to the cloud?” or “How can we ensure our data remains secure?” Finally, use The Power of No: “Would it be a ridiculous idea to explore a hybrid model?”

19.3.2.2: Scenario 2: Negotiating adopting a new software development methodology.

Recognize the team's attachment to the current methodology with Tactical Empathy: "I see that you've been very comfortable with our current Agile practices." When a team member says, "I don't think this new methodology will work," respond with Mirroring: "Won't work?" When they elaborate, continue mirroring their key phrases to draw more details. For example, if the team member says, "Yes, it's too different from what we're used to," respond, "Too different from what we're used to?" When they add, "Yes, we have established routines and this new approach will disrupt them," say, "Established routines and disruptions?" This might lead to, "Yes, we're worried that it will slow down our progress and make us less efficient initially." Label their concerns with, "It sounds like you're worried that this change will lead to inefficiencies and slow down our progress." Perform an Accusation Audit, stating, "You might think this change is unnecessary and disruptive." Use Calibrated Questions like, "What steps can we take to ensure this new methodology integrates smoothly with our existing processes?" or "How do you see this change improving our current workflow?" Use The Power of No with, "Is it a terrible idea to pilot this methodology on a smaller project first?"

19.3.2.3: Scenario 3: Negotiating deadlines with the product management team.

Show Tactical Empathy by understanding the pressure the product management team might be under to deliver on time: "It sounds like you're under a lot of pressure to meet these deadlines." Use Mirroring when a product manager says, "We need this feature by the end of the month," by responding, "By the end of the month?" When they say, "Yes, the marketing team has already planned a campaign around it," you might respond with, "Planned a campaign?" They might elaborate, "Yes, and if we miss the deadline,

it will throw everything off.” Follow up with, “Throw everything off?” leading to, “Yes, it will affect our quarterly targets.” Label this by saying, “It seems like you’re worried about how delays might affect the quarterly targets.” Perform an Accusation Audit with, “You might think the engineering team is slow and unresponsive to deadlines.” Use Calibrated Questions such as, “What can we do to ensure the most critical features are delivered on time?” or “How can we prioritize tasks to meet your needs without compromising quality?” Use The Power of No: “Would it be unreasonable to extend the deadline by two weeks to ensure the feature is fully tested?”

19.3.2.4: Scenario 4: Convincing a team to refactor legacy code.

Show Tactical Empathy by understanding the team’s attachment to the existing codebase and the fear of extensive changes: “It sounds like you’re concerned about the potential risks of changing the existing codebase.” Use Mirroring when a team member says, “The legacy code works fine as it is,” by responding, “Works fine as it is?” When they say, “Yes, we haven’t had any major issues with it,” respond with, “No major issues?” When they add, “Right, and changing it might introduce new problems,” you could say, “Introduce new problems?” leading to, “Yes, and we’re not sure if the benefits are worth the risks.” Label this by saying, “It sounds like you’re worried that refactoring might introduce new problems and the benefits might not be worth the risks.” Perform an Accusation Audit by stating, “You might think this refactor is just going to be a huge time sink without much benefit.” Use Calibrated Questions such as, “What are the biggest risks you see with refactoring the legacy code?” or “How can we ensure the refactor improves performance without disrupting current functionality?” Use The Power of No: “Would it be a bad idea to refactor the code in small, manageable sections?”

19.3.2.5: Scenario 5: Gaining team buy-in for adopting a new technology stack.

Show Tactical Empathy by understanding the team's hesitation towards learning and integrating new technologies: "It sounds like you're concerned about the learning curve associated with this new technology." Use Mirroring when a team member says, "I don't see the need for this change," by responding, "Don't see the need?" When they say, "Yes, our current stack is working fine for us," respond with, "Working fine for us?" When they add, "Yes, we're familiar with it, and it meets our needs," you could say, "Meets our needs?" leading to, "Yes, switching to something new would be a hassle." Label this by saying, "It sounds like you're worried about the hassle of switching to a new technology, and you feel the current stack meets our needs." Perform an Accusation Audit by stating, "You might think this new technology will be too complex and time-consuming to adopt." Use Calibrated Questions such as, "What are the main challenges you anticipate with adopting this new technology?" or "How can we support the team to make the transition smoother?" Use The Power of No: "Would it be a terrible idea to start with a small, non-critical project to evaluate the benefits of the new stack?"

19.3.2.6: Scenario 6: Managing unrealistic project timelines imposed by senior management.

Show Tactical Empathy by understanding senior management's pressures regarding market competition and business goals: "It sounds like there's significant pressure to meet market demands quickly." Use Mirroring when a senior manager says, "We need the project completed in three months," by responding, "Three months?" When they say, "Yes, any longer, and we risk falling behind our competitors," they respond with, "Falling behind competitors?" When they add, "Yes, we must launch before they do," you could say, "Launch before they do?" leading to, "Yes, to capture

the market share we're targeting." Label this by saying, "It sounds like you're worried about losing market share if we take longer." Perform an Accusation Audit, stating, "You might think we're not committed to meeting the company's goals." Use Calibrated Questions such as, "How can we ensure that the quality of the project is not compromised while aiming for a faster timeline?" or "What are the most critical features that need to be prioritized to meet this deadline?" Use The Power of No: "How am I supposed to deliver a high-quality, fully-tested product in three months with the current resources and constraints?"

The last example, "How am I supposed to do that?" is a calibrated question designed to make the other party reconsider the feasibility of their demand and encourage a more realistic and collaborative discussion about what can be achieved. This approach helps negotiate more reasonable timelines or secure additional resources to meet tight deadlines.

19.4: Questions to Consider

- *In what ways can understanding your audience's needs and motivations enhance your presentations or meetings?*
- *What challenges do you face when delivering messages that contain bad news or require a significant change? How can the “Houston, We Have a Problem” storyline help in these situations?*
- *Reflect on a recent scenario where clear and concise messaging could have changed the outcome. What would you do differently now?*
- *How often do you use stories and analogies to explain technical concepts? Can you think of an analogy that effectively illustrates a complex idea you often work with?*
- *What techniques from “Radical Candor” could you use to improve the feedback culture within your team?*
- *How might tactical empathy improve your negotiation skills in IT project discussions?*
- *Think of a time when mirroring a statement could have helped you gather more information or clarity during a discussion. How might you apply this technique in the future?*
- *Which calibrated question could you have used to guide the discussion more effectively in a recent challenging conversation?*
- *How do you prepare for important communications or negotiations? What practices ensure you are clear and confident in your delivery?*
- *How do accusation audits and labeling emotions affect your interactions with project teams or stakeholders?*
- *What are your strategies for handling unrealistic expectations from upper management or clients? How could negotiation techniques from “Never Split the Difference” help in these situations?*

20: Understanding Product Development

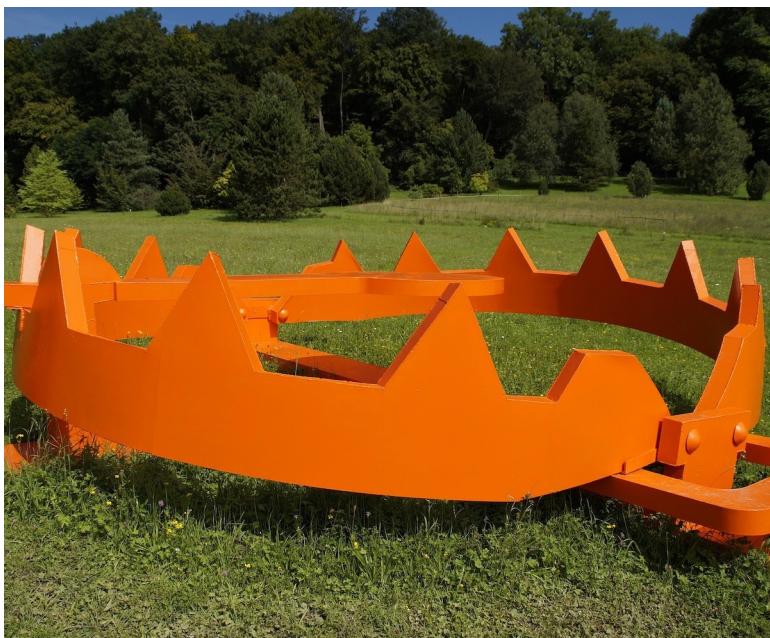


image by m w from pixabay

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

KEY POINTS:

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

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

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

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

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

- ment Creates Real Value¹” by Melissa Perri, and
- “The Discipline of Market Leader²” by Michael Treacy and Fred Wiersema.

Both of these sources provide several things for architects:

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

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

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

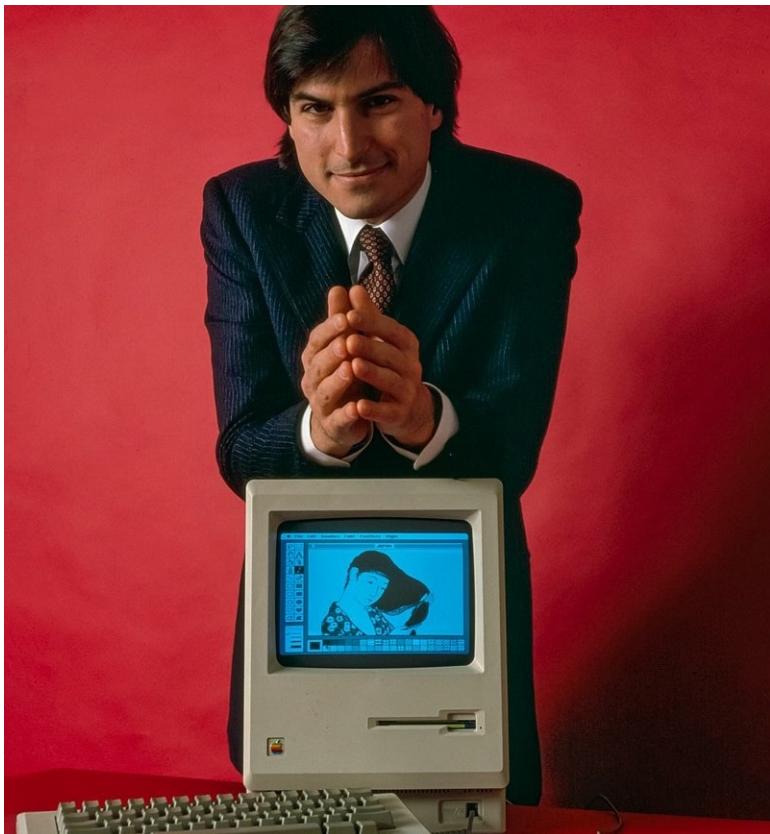


image by wikimedia commons

20.1: 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

20.1.1: 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
	Acquisition measure when someone first starts using your product or service	Number of new signups or qualified leads
		Customer acquisition cost (CAC)
	Activation 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
	Engagement measure how (and how often) users interact with your product	Monthly, weekly, daily active users (MAU, WAU, DAU)
		Stickiness (DAU/MAU)
		Feature usage
	Retention gauge how many of your users return to your product over a certain period of time	Retention rate
		Churn rate
		Customer lifetime value (CLV)
	Monetization 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.

20.1.2: 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.

20.1.3: 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.

20.1.3.1: The Mini-CEO

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

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

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



image by istock

20.1.3.2: The Former Project Manager

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

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

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

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

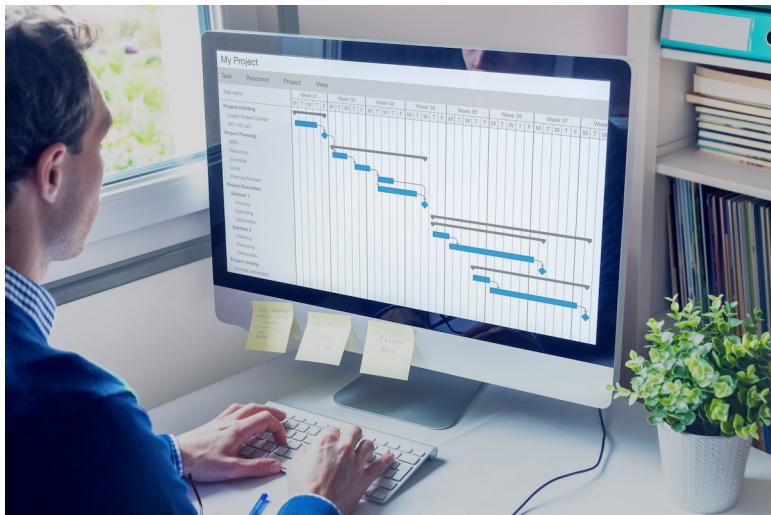


image by istock

20.1.3.3: 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**³:

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



image by istock

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

20.2: The Discipline of Market Leader

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

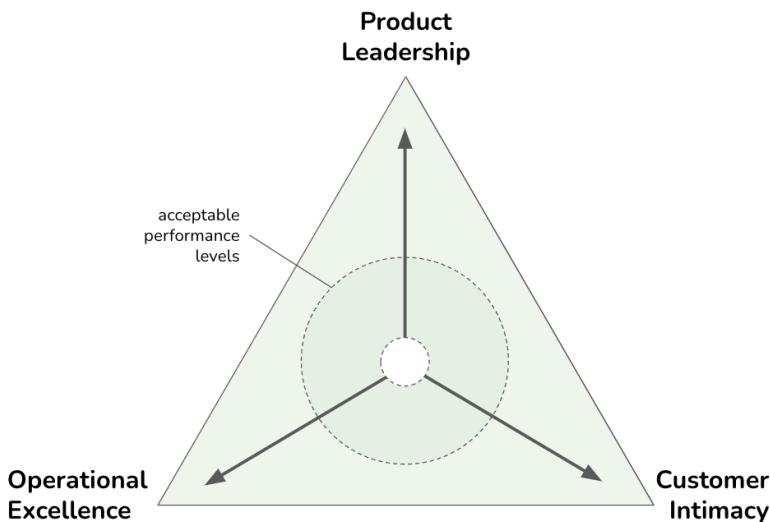


Figure 2: The Discipline of Market Leader model postulates that any successful business needs to maintain at least “acceptable” levels of performance in each of the three dimensions (operational excellence, product leadership, and customer intimacy) but would need to choose one of them to become a market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

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

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

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

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

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

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

I found the model helpful in two ways:

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

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

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

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

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

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

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

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

20.3: Product Operations

Another product concept relevant to architectural practice is **Product Operations**. Melissa Perri and Denise Tilles provide an excellent overview of Product Operations in their book⁵. 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

⁵<https://www.productoperations.com/>

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

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

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

20.3.1: 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**.

20.3.2: 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**.

20.3.3: 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.

20.3.4: Product Operations and Architecture Practice

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

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

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

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

20.4: 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?*

21: Architecture Governance: Nudge, Taxation, Mandates



image by nonbirinonko from pixabay

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

KEY POINTS:

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

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

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

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

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

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

21.1: Nudging

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

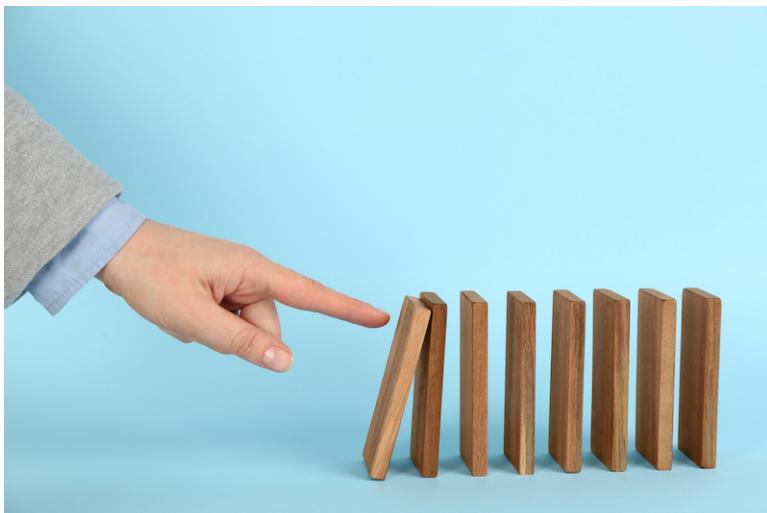


image by istock

A nudge can take many forms, such as a slight change in the environment, a gentle reminder, 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**¹
- **Highlighting** bad quality software on a **Data Foundation** 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 **Data Foundation** tools to **highlight areas and issues** we wanted (nudged) people to improve. The **People Foundation**

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

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

21.2: Taxation (Economic Incentives)

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



image by steve buissinne from pixabay

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

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

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

21.3: Mandates and Bans

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

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



image by tumisu from pixabay

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

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

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

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

21.4: 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?*

22: Economic Modeling: ROI and Financial Options



image by nattanan kanchanaprat from pixabay

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

KEY POINTS:

- Architects are frequently asked about the (economic) value of architecture or technology investments.
- Answering this question is a crucial skill for any senior architect. 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.

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

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

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

As financial and risk modeling is essential in any organization, architects frequently need to answer questions about the (economic) value of technology investments and architecture. Answering this question is a crucial skill for any senior architect. 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

22.1: The Return-on-Investment Metaphor

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

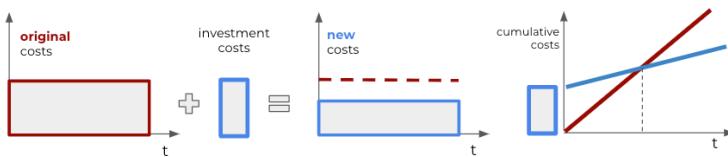


Figure 1: An illustration of the ROI metaphor. Investment leads to lower costs or higher value. It takes some time to reach a break-even point when an additional value has compensated for the investment. After the break-even point, we earn more than without the investment.

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

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

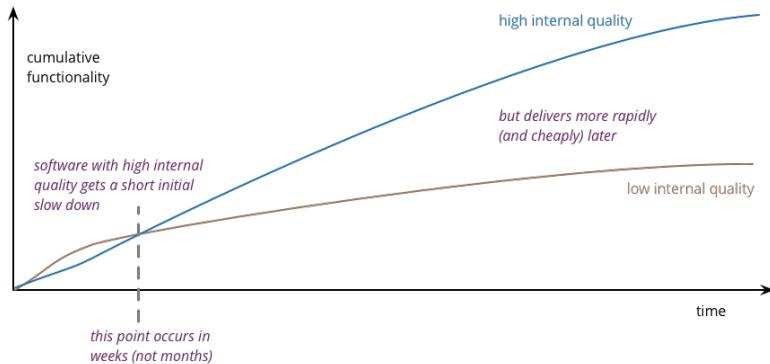


Figure 2: Software with high internal quality gets a short initial slowdown but delivers more rapidly and cheaply later (source martinfowler.com/articles/is-quality-worth-cost.html).

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.

22.2: 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](#)². 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).

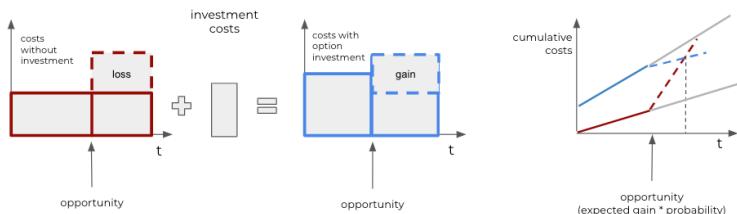


Figure 3: An illustration of the financial option metaphor. Options have a price, leading to higher initial costs. However, if an opportunity can generate more value, we gain additional profit (or lose it if we do not invest).

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

²https://en.wikipedia.org/wiki/Black%20Scholes_model

(virtual) servers can be ordered later at a known unit cost.

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

22.3: A Communication Framework

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

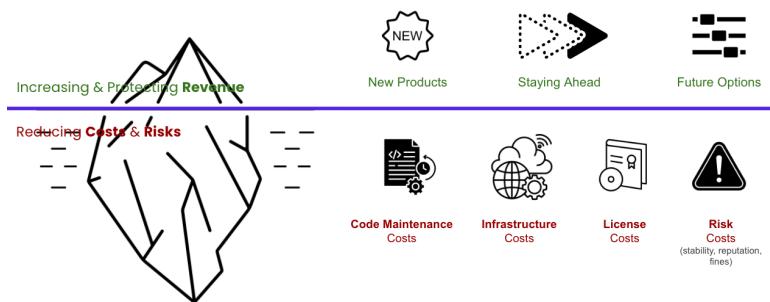


Figure 4: A framework for discussing investments and options.

I separate the value of investments in two buckets:

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

Increasing and protecting revenue investments have three forms:

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

product as the rest of the world moves on. As the saying goes, “**It takes all the running you can do to keep in the same place.**” For instance, you must keep essential features in parity with the competition, your system must comply with changes in regulations, and your UX must be modern.

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

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

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

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

- **Investments in reducing risk costs.** When your system is down, your business is disrupted, and you lose revenue. According to diverse studies³, the average cost of downtime ranges from \$2,300 to \$9,000 per minute. You must invest in keeping your system reliable and secure to avoid losing revenue and disrupting your business. While the benefits of these types of investments are huge, the challenge with building the business case for this investment is that a reliable system will only create a few incidents, making it less tangible for many stakeholders to understand the importance of continuing such investments. Or, as noted by Repenning and Sterman “[Nobody Ever Gets Credit for Fixing Problems that Never Happened](#)⁴”.

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

⁴https://web.mit.edu/nelsonr/www/CMR_Getting_Quality_v1.0.html

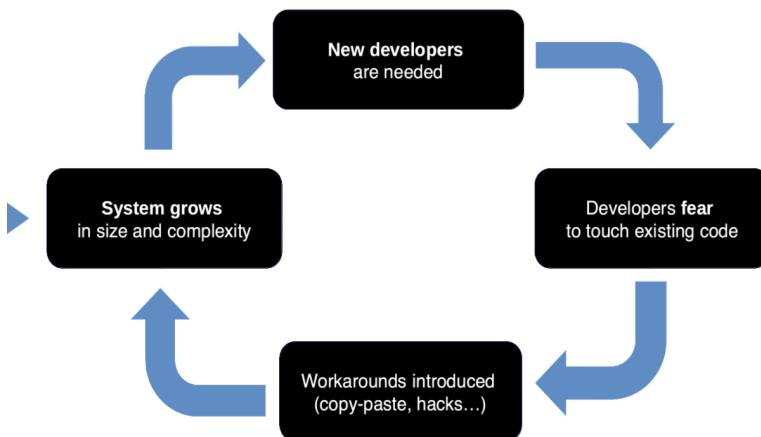


Figure 5: A downward spiral of poorly maintainable code. As such systems grow in size and complexity, more developers are needed to maintain them. If the system is not easy to maintain, people will avoid touching code as they can easily break it. This situation will lead to a workaround (such as copying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

22.4: Questions to Consider

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

23: Decision Intelligence in IT Architecture



image by istock

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

KEY POINTS:

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

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

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

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

¹https://en.wikipedia.org/wiki/Cassie_Kozyrkov

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

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

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

- By **creating mechanisms** for teams to make better decisions (e.g., [advisory forums⁴](#)).
- By **creating options⁵** for teams to make decisions later.

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

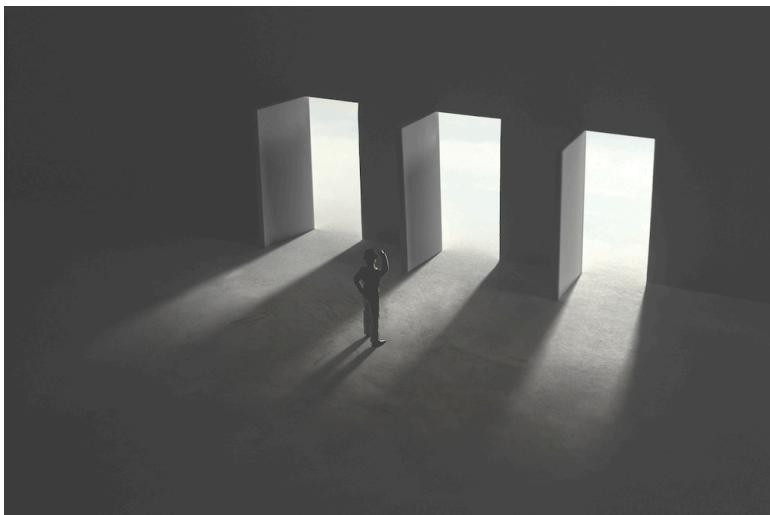


image by istock

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

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

23.1: Basics of Decision-Making

Let's start with some basics: defining decisions, outcomes, and goals.

23.1.1: 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](#)⁶. 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.

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

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

23.1.2: 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](#)⁷). 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).

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

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

23.1.3: 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](#)⁸” concept (also known as the value of perfect information) in decision analysis. This nifty idea helps you 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

⁸https://en.wikipedia.org/wiki/Value_of_information

you balance the effort you put into decision-making with the decision's actual importance.

For example, deciding on the best public cloud provider is like choosing a life partner. 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!

23.2: 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.

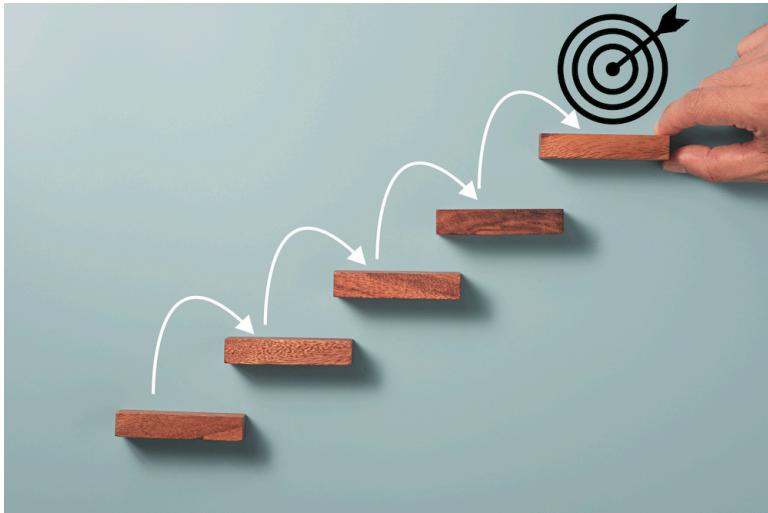


image by istock

23.2.1: 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.

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.

23.2.2: 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.

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**⁹”—supporting a limited set of technologies and making it tougher to stray into uncharted territory.

⁹<https://engineering.spotify.com/2020/08/how-we-use-golden-paths-to-solve-fragmentation-in-our-software-ecosystem/>

It's like saying, "Sure, you can build with LEGO, but maybe let's stick to this one box instead of the entire toy store."

So remember, setting those golden paths and constraints isn't about being a killjoy. It's about keeping everyone aligned and avoiding a tech Tower of Babel.

23.2.3: 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.

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.



image by istock

23.3: Decision-Making Complexity

Some decisions are easier to make than others. Kozyrkov identifies 14 factors that can make decision-making more or less complex:

23.3.1: 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.

23.3.2: 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.

23.3.3: 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.

23.3.4: 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.

23.3.5: 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.

23.3.6: 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.

23.3.7: 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.

23.3.8: 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.

23.3.9: 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.

23.3.10: 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.

23.3.11: 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.

23.3.12: 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.

23.3.13: 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.

23.3.14: 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.

23.4: Decision-Making With Data and Tools

Decision-making has evolved beyond pen and paper, with data playing a crucial role in modern methods. Data, like the one I use in [Data Foundation](#), while visually appealing and powerful when used correctly, is **only a tool to assist in making informed decisions**. It's a means to an impactful end, but **without purposeful application, data is ineffective**.



image by istock

23.4.1: Remember that data has limitations

Just as not everything written in a book is true, data can be **misleading or incomplete**. It's a collection of information recorded by humans, subject to errors and omissions.

For instance, in 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.

23.4.2: Chose the right tool for the job

Cassie Kozyrkov breaks down the techniques for analyzing data into three snazzy **groups**¹⁰:

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 or 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.

¹⁰<https://kozyrkov.medium.com/what-on-earth-is-data-science-eb1237d8cb37>

23.4.3: 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.

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.

23.4.4: Be in the driving seat

Just staring at data and crunching numbers isn't decision-making. As the decision maker, your job is to set the stage, choose the relevant topics, frame the right questions, and guide the analysis like a maestro conducting an orchestra.

As a decision-maker, it's crucial to **ask the right questions**, and figure out **which decisions are worth your brainpower**. Once you've identified those 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.

23.5: Questions to Consider

- *How do you typically approach decision-making in your professional role, and in what ways could you incorporate the principles of decision intelligence to enhance your decision-making process?*
- *Have you observed instances where excessive 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?*
- *Can you identify any habits or emotional factors contributing to your indecisiveness? What strategies can you employ to overcome these challenges?*
- *How do you use data in your decision-making process? Are there instances where data has misled your decisions, and how can you safeguard against this in the future?*

24: The Human Side of Decision-Making



image by istock

IN THIS SECTION, YOU WILL: Learn the basic human factors influencing decision-making.

KEY POINTS:

- Decision-making is a human activity subject to human biases and limitations.
- Fundamental biases influencing decision-making include outcome, hindsight, and confirmation biases.
- Human intuition plays a vital role in decision-making.
- Group decision-making offers significant advantages but increases complexity as it requires higher decision-making skills from each member.

Cassie Kozyrkov¹, in her [posts](#)² and [online lessons](#)³ about design intelligence, often reminds us that decision-making is a uniquely human sport. It's a wild mix of brilliance, bias, and blunders. Just like how having a gourmet kitchen doesn't make you a master chef, having the best [data and IT tools](#) won't magically conjure up good decisions. IT architects need to remember that every decision is flavored by the quirks and quibbles of the human mind.

¹https://en.wikipedia.org/wiki/Cassie_Kozyrkov

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

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

24.1: Biases and Limitations

Fundamental biases influencing decision-making include outcome, hindsight, and confirmation biases.

24.1.1: Outcome Bias

The big trap in decision-making is falling into the **outcome bias** pit, where you **obsess over the results** rather than focusing on how you made the decision in the first place. If you trip over an unlucky result and think you picked the wrong option, you're learning how to fail with flair. This faulty thinking could make you make even wackier choices in the future.



image by istock

A decision and its outcome are like two different TV show episodes. The outcome is just what happens after the decision's cliffhanger. You can make a totally solid decision and still end up with a plot twist no one saw coming. Remember, outcomes are a cocktail mixed with decision-making, randomness, and a **splash of luck**. And luck, as we all know, is that elusive ingredient we can't control, often playing a starring role in our most complex dramas.

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.

Example 6: Hiring Decisions Based on Project Success

- A software engineer is hired because they were part of a highly successful project at their previous company. The project's success was primarily due to market conditions and team effort rather than the individual's contributions.
- The hiring decision is heavily influenced by the success of the previous project, potentially overlooking the individual's actual skills and contributions. This could result in hiring someone who may not perform as expected in different circumstances, highlighting the risks of evaluating individual capabilities based on outcomes rather than detailed assessment.

It's essential for teams to critically evaluate their methodologies and ensure that success is attributed to sound practices rather than favorable outcomes alone.

24.1.2: Hindsight Bias

Looking back at decisions can be as tricky as explaining a magic trick once you know the secret, thanks to **hindsight bias**. Everything seems obvious in hindsight, even though it was as clear as mud at the time. The real way to judge a decision is to dig into the info and context available when it was made—like a detective piecing together clues.

Think about what the decision-maker considered, how they played detective gathering info, and where they got their facts. Also, check if they collected enough intel for the decision's importance or were winging it.

If you don't jot down this process, it's like trying to remember a dream—you'll miss many details and be left only with a vague feeling. This makes it tough to judge the quality of your decisions

or anyone else's, stalling your growth as a decision-making wizard. Write down your decision-making process. This helps you determine if luck was messing with you or if your skills were on point.



image by istock

Hindsight bias in IT can lead to an oversimplified understanding of past events and decisions, creating an illusion of predictability. It is crucial to recognize the context and constraints under which decisions were made to learn accurately from past experiences and improve future practices.

Example 1: Technology Stack Decision

- A particular technology stack is chosen for a project, but it later turns out to be ill-suited, causing significant rework.
- After the problems become apparent, developers and stakeholders might claim that it was clear from the beginning that this technology stack was not a good choice. They might forget that at the time of selection, the decision was made

based on the best available information and the perceived advantages of the stack.

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.

24.1.3: 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.

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.



image by istock

Beating confirmation bias is like prepping for a magic show: you need a plan before the curtain goes up. **Set clear objectives** before you peek at the data. Consider what the data should mean to you beforehand so you don't get dazzled by surprising plot twists. This way, you can make genuinely data-driven decisions instead of falling into the trap of your brain's sneaky biases.

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.

24.1.4: Other Human Limitations

In a classic behavioral economics study, researchers showed that **how you say something** can be more magical than a rabbit out of a hat. They gave decision-makers the same facts but with some wordplay—different wording. Despite the identical information, decisions did a Houdini act and varied wildly. A tweak in phrasing or tossing in unrelated details can make people's choices wobble like a tightrope walker in a windstorm. Our brains are suckers for cognitive biases and illusions, even when the cold, hard facts are staring us in the face.



image by istock

This means that dealing with data isn't as objective as we like to think. How we mentally wrestle with data matters a lot. We may aim to use data to become more objective, but if we're not careful, it can pump up our pre-existing beliefs like a bouncy castle. Instead of uncovering new insights, we end up with our same old convictions, sabotaging our quest for objectivity and learning.

And let's be honest—your decision-making skills aren't precisely

Olympic-level when you're **sleep-deprived, hungry, stressed**, or feeling the heat. These biological and emotional states can affect your ability to make top-notch decisions. Believing that sheer willpower or a PhD in decision-making will always lead to the best outcomes is like thinking you can win a marathon in flip-flops.

A jaw-dropping example is in the legal system: studies show that judges can hand out different sentences before and after lunch. Even these wise, experienced folks, whose decisions shape lives, can be swayed by something as simple as a rumbling stomach. This should be a big, flashing neon sign warning us about the limits and quirks of our decision-making processes. So next time you're about to make a big decision, grab a snack and a nap first!

24.2: Indecisiveness

We frequently fall into the indecisiveness trap. Why? Well, people can be indecisive for all sorts of amusing reasons.

24.2.1: Bad Habits

First up, bad habits. 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 ice cream-less. Delaying, postponing, or deprioritizing the decision-making process is an implicit choice.

For instance, if a company can't decide on using one consolidated tech stack for its systems, it ends up with a tech landscape messier than a toddler's playroom. That implicit decision comes with all the extra costs and complexities you'd expect.

24.2.2: 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.

For instance, sometimes people treat IT architects like decision-making oracles, asking them to approve every trivial change (like upgrading a minor library version). This is like asking the CEO to decide on the office snack rotation—it's a sure way to waste their strategic brainpower.

24.2.3: 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.

24.3: Intuition

Human intuition plays a vital role in decision-making. Robert Glass provided one of the best definitions of intuitions, describing it as a function of our mind that allows it to access a **rich fund of historically gleaned information** we are not necessarily aware we possess by a **method we do not understand** (Glass, 2006; page 125)⁴. Our unawareness of such knowledge does not mean we cannot use it.

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.

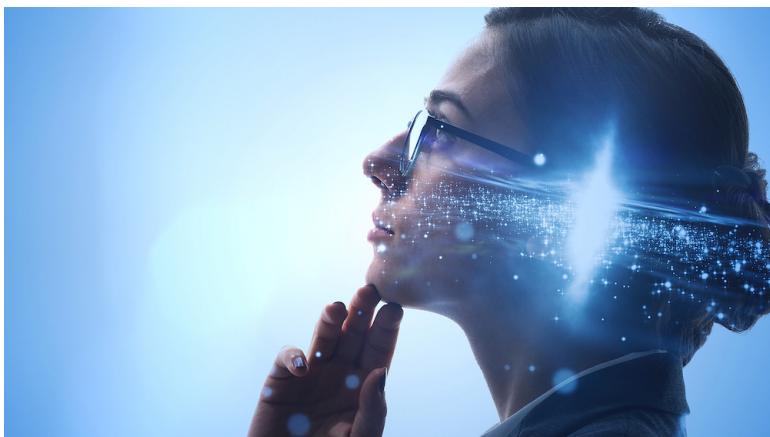


image by istock

Intuition is especially appropriate under certain conditions:

⁴<https://www.amazon.com/Software-Creativity-2-0-Robert-Glass/dp/0977213315>

- **Time Pressure:** When time is limited and a detailed analysis isn't feasible, intuition can guide you.
- **Expertise:** If you have experience in a particular area, relying on intuition makes sense, as you've likely faced similar decisions before. In contrast, in unfamiliar contexts, intuition may not be reliable.
- **Unstructured Decisions:** Intuition can be valuable for decisions that lack a clear framework, like judging the quality of art. Expertise in the relevant field enhances the effectiveness of intuitive judgments.

Conversely, you should avoid relying on intuition too much in situations where more effort is warranted, including those with ample time, high importance, lack of expertise, and a structured decision-making process.

Intuition in IT can be a powerful tool when 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.

24.3.0.1: Good Example: Intuitive Debugging

Scenario: A seasoned software engineer is working on a complex system that suddenly starts behaving unexpectedly. The logs provide little information, and initial debugging efforts do not yield 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.

24.3.0.2: Bad Example: Intuitive Decision on Technology Stack

Scenario: A new project is starting, and the team needs to decide on the technology stack. The team lead has a strong intuitive preference for a specific new programming language and framework 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.

24.4: Group Decision-Making Dynamics

Effective decision-making often involves recognizing that **you might not be the sole decision-maker**. In organizations, it's crucial to identify the actual decision-makers and understand how decision responsibility is distributed among them. Mastering this skill is essential for navigating organizational decision-making processes. It's important to question who really has the final say in decisions. In many cases, decision-making is more complex than it appears.



image by istock

Group decision-making offers significant advantages. While you might believe you have the best solutions, incorporating diverse perspectives can help cover your blind spots. Multiple decision-makers can counterbalance an individual's extreme tendencies and compensate for human limitations like fatigue.

While group decision-making might sometimes constrain individual creativity, it also provides **safeguards against poor decisions** and **aligns individual motives with the organization's goals** (see the principal-agent problem in the next section). Having several independent decision-makers can align individual incentives with

the organization's needs, addressing this problem.

However, group decision-making isn't perfect. It **increases complexity** as it requires higher decision-making skills from each member. True **collaboration in decision-making** is more challenging than individual decision-making. It also tends to **slow down the decision process**.

Moreover, the benefits of group decision-making, like balancing individual biases, **rely on the independence of the decision-makers**. If everyone is in the same room, independence can be compromised by factors like **charisma or status**, potentially allowing the loudest voice to dominate rather than the wisest.

Group settings can also **devolve into social exercises**, where **personal ego overshadows open-mindedness** to new information. 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.

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:

24.4.1: 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.

24.4.2: 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.

24.4.3: 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.

24.4.4: 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.

24.5: Questions to Consider

- *How do your personal biases, such as outcome, hindsight, and confirmation bias, influence your decision-making process? Can you identify a recent decision where these biases might have played a role?*
- *Reflect on a situation where the outcome was bad, but the quality of your decision-making process was solid. How did you respond to this outcome, and what lessons did you learn?*
- *In what ways do you think hindsight bias has affected your ability to evaluate past decisions accurately? Can you think of a decision that seemed obvious in retrospect but was unclear at the time?*
- *Consider a decision you made recently. Did you document the decision-making process? If not, how could documenting this process help you in evaluating your decisions more effectively in the future?*
- *How does confirmation bias impact your interpretation of new information? Can you recall an instance where you ignored or misinterpreted data to fit your pre-existing beliefs?*
- *Think about a decision where changing your perspective or how information was presented (e.g., through different wording) might have led you to a different conclusion. How does this realization affect your approach to decision-making?*
- *Reflect on a time when your physical or emotional state might have influenced a decision. What does this tell you about the importance of being aware of your condition when making decisions?*
- *Consider a decision where intuition played a significant role. Was the decision effective, and would you rely on intuition under similar circumstances in the future?*
- *How do you balance the benefits of group decision-making with its challenges, such as social dynamics and the diffusion of responsibility?*

Part IV: Wrapping Up

25: Summary



image by 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 function to stay well-connected to all levels of an organization and led by data.
- In this section, I summarize the key points from the playbook.

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

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

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

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

- **Data Foundation**,

- People Foundation,
- Architecture Activities Platform.

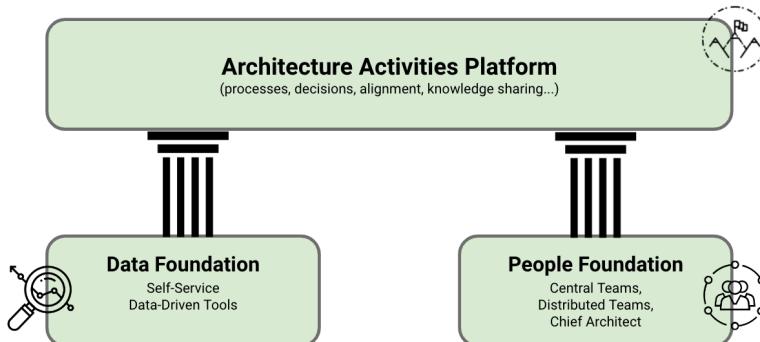


Figure 1: The structure of Grounded Architecture.

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

The *People Foundation* is another essential element of Grounded Architecture. A strong network of people doing architecture across the organization is crucial to ensure that architecture function has any tangible impact.

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

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

- Being an Architect:
 - Architects as Superglue

- Skills
- Impact
- Leadership
- Architects' Career Paths

- Doing Architecture: Inspirations:

- Culture Map: Architects' Culture Mindfield Compass
- Managing Organization Complexity: Six Simple Rules
- Product Development and The Build Trap
- Architecture Governance: Mandates, Taxation, Nudge
- Economic Modeling: ROI and Financial Options
- Decision Intelligence in IT Architecture
- The Human Side of Decision-Making

When Grounded Architecture is in place, it can have a significant positive impact on the functioning of an organization:

- Enable Execution of Architecture Function At Scale,
- Increase the Quality of Decision-Making with Data,
- Maximize Organizational Alignment,
- Maximize Organizational Learning, and
- Increase Architecture Function Adaptivity.

While you may borrow ideas from others, every organization is different, and your approach must adapt to your context. When forming architecture functions, I always advise not to forget these two pieces of advice from Gregor Hohpe¹:

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

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

26: Cheat Sheet

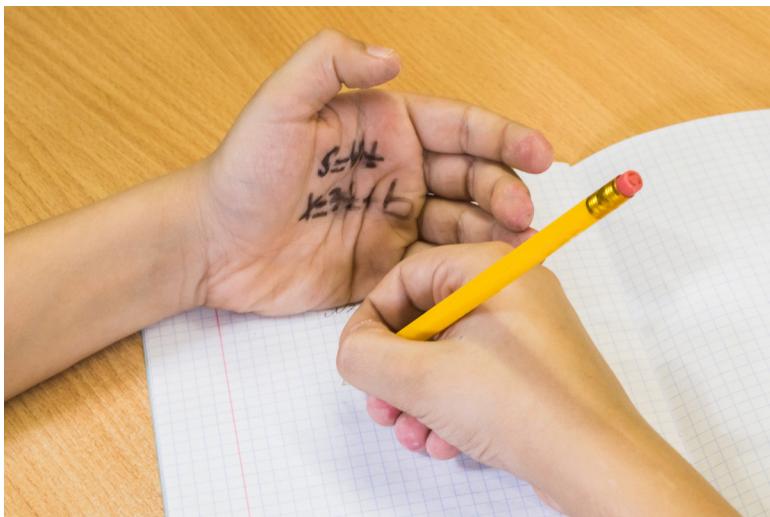


image by istock

IN THIS SECTION, YOU WILL: Get a cheatsheet with all key points from in all sections.

26.1: Introductions

26.1.1: Introduction

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

26.1.2: Context: Fast-Moving Global Organizations

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

26.1.3: Evolution of Architecture: Embracing Adaptability, Scalability, and Data-Driven Decisions

- I identified the following needs that an architecture function should support: Executing At Scale, Adaptivity, Improving

the Quality of Decision-Making with Data, and Maximizing Organizational Alignment & Learning.

26.2: Grounded Architecture: Introduction

26.2.1: Data Foundation

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

26.2.2: People Foundation

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

26.2.3: Architecture Activities Platform

- The Architecture Activities Platform is a system of processes and agreements enabling architects to do everything architec-

ture typically does, leveraging Data and People Foundations to create a data-informed, organization-wide impact.

- Examples of activities include: supporting teams in their daily work; tracking tech debt, defining tech debt reduction programs; performing technical due diligence; standardizing processes and documentation; defining cloud, data, and platform strategies.

26.2.4: Transforming Organizations with Grounded Architecture

- When a Grounded Architecture structure 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.

26.3: Being Architect: Introduction

26.3.1: Architects as Superglue

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

26.3.2: Skills

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

26.3.3: Impact

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

26.3.4: Leadership

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

26.3.5: Architects' Career Paths: Raising the Bar

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

26.4: Doing Architecture: Introduction

26.4.1: The Culture Map: Architects' Culture Compass

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

26.4.2: Managing Organizational Complexity: Six Simple Rules

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

26.4.3: Effortless Architecture

- 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.

26.4.4: Understanding Product Development

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

26.4.5: Architecture Governance: Nudge, Taxation, Mandates

- 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.

26.4.6: Economic Modeling: ROI and Financial Options

- Architects are frequently asked about the (economic) value of architecture or technology investments.
- Answering this question is a crucial skill for any senior architect. 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.

26.4.7: Decision Intelligence in IT Architecture

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

26.4.8: The Human Side of Decision-Making

- Decision-making is a human activity subject to human biases and limitations.
- Fundamental biases influencing decision-making include outcome, hindsight, and confirmation biases.
- Human intuition plays a vital role in decision-making.
- Group decision-making offers significant advantages but increases complexity as it requires higher decision-making skills from each member.

Part V: To Probe Further

27: Bookshelf

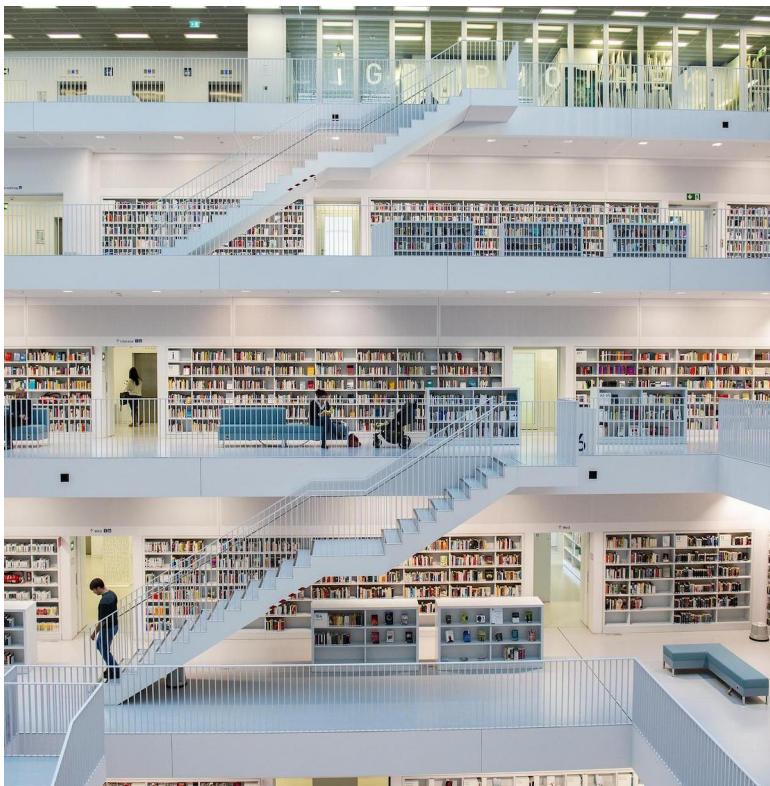


image by oli gotting from pixabay

IN THIS SECTION, YOU WILL: Get an overview of the background work to probe further, linking several external resources inspiring my work.

27.1: Introduction



The Software Architect Elevator: Redefining the Architect's Role by Gregor Hohpe defines architects as people that can fill a void in large enterprises: they work and communicate closely with technical staff on projects but are also able to convey technical topics to upper management without losing the essence of the message. Conversely, they understand the company's business strategy and can translate it into technical decisions that support it.

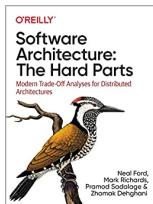


SE Radio: On Architecture¹: for those interested in IT Architecture, I've created a curated Spotify playlist of Software Engineering Radio Episodes focusing on Architecture.

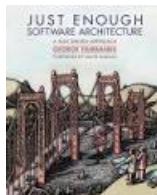
¹<https://open.spotify.com/playlist/7GVD86edcILnVPjsZFTy28?si=f44d0bef360e4818>



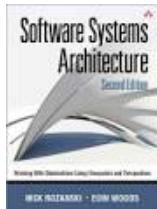
Software Architecture for Developers by Simon Brown is a practical, pragmatic, and lightweight software architecture guide specifically aimed at software developers.



Software Architecture: The Hard Parts is structured as a narrative about a team breaking down a faulty, outdated monolithic application into a modern microservices-based architecture. The authors compare different aspects of how a monolithic architecture might have been written to do something in the past, then how modern microservice architecture could do the same thing today, offering advice and approaches for practical trade-off analysis when refactoring a large monolith app.



Just Enough Software Architecture by George Fairbanks is an approachable and comprehensive book.

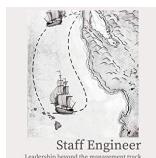


Software Systems Architecture by Nick and Eóin is another practitioner-oriented guide to designing and implementing effective architectures for information systems.

27.2: Career Development



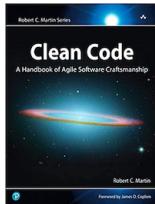
The Staff Engineer's Path by Tanya Reilly. Similar to my view of architects, the staff engineer's path allows engineers to contribute at a high level as role models, driving big projects, determining technical strategy, and raising everyone's skills.



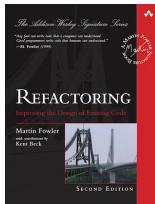
Staff Engineer: technical leadership beyond the management track by Will Larson defined technical leadership beyond the management track. I share many of the views presented in this book regarding the development and skills of architects.

27.3: Hard Skills

This section is heavily influenced by Gregor Hohpe's article [The Architect's Path - Part 2 - Bookshelf](#)²



Clean Code by Bob Martin discusses good code, and good code is clean. The basics of naming, functions that do one thing well, and formatting.

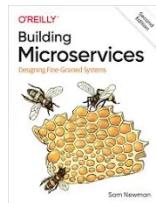


Refactoring: Improving the Design of Existing Code by Martin Fowler. Good software evolves, gains entropy, and is then restructured. Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior.

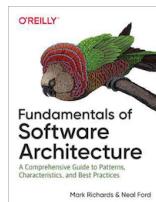
²<https://architectelevator.com/architecture/architect-bookshelf/>



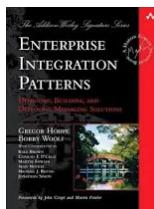
Design Patterns by Gamma, Helm, Johnson, and Vlissides. Design patterns help us make balanced decisions on the design of our code.



Building Microservices (2nd Edition) by Sam Newman is a book about architectural trade-offs and considerations in distributed system design.



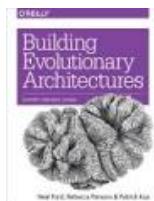
Fundamentals of Software Architecture by Neal and Mark covers soft skills, modularity, component-based thinking, and architectural styles.



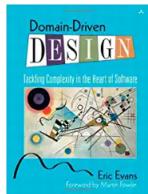
Enterprise Integration Patterns for anyone trying to connect systems without coupling them too tightly.



Pattern-Oriented Software Architecture: one of the most comprehensive references for distributed system design.



Building Evolutionary Architectures describes how fitness functions can guide architectural change over time.



Domain-Driven Design by Eric Evans promotes the idea that to develop software for a complex domain, we need to build Ubiquitous Language that embeds domain terminology into the software systems that we build. DDD stresses defining models in software and evolving them during the software product's life.



Release It! (2nd Edition) by Mike Nygard's is about architecture stability decisions and how to build “cynical” software. Besides the well-known stability patterns like circuit breakers and bulkheads, the book introduces foundational knowledge about running systems in production, from processes to network to security, also covering more process-oriented questions like deployments or handling security.



Designing Data Intensive Systems by Martin Kleppmann is a mini-encyclopedia of modern data engineering. The book lays down the principles of current distributed big data systems. It covers replication, partitioning, linearizability, locking, write skew, phantoms, transactions, event logs, and more.



Thinking in Systems by Donella Meadows describes a system as more than the sum of its parts. It may exhibit adaptive, dynamic, goal-seeking, self-preserving, and sometimes evolutionary behavior.



Don't Make Me Think, Revisited: a common sense approach to Web usability by Steve Krug provides a practical guide for understanding web usability and user experience.

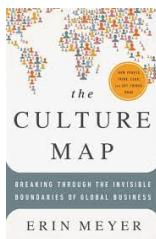


System Design Interview (Volume 1 & 2) by Alex Xu. A “real-world” systems design book is not just for preparing for the systems design interview but also for strengthening your systems design muscle for the day-to-day architectural work.

27.4: Soft Skills



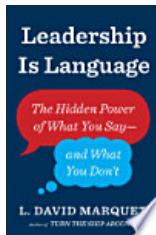
Team Topologies by Manuel Pais and Matthew Skelton describes four fundamental topologies (stream-aligned teams, enabling teams, complicated subsystem teams, and platform teams), and three fundamental interaction modes (collaboration, X-as-a-Service, and facilitation).



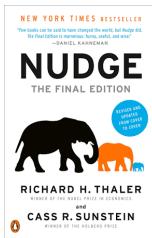
The Culture Map: Decoding How People Think, Lead, and Get Things Done Across Cultures: provides a framework for handling intercultural differences in business and illustrates how different cultures perceive the world. Awareness of intercultural differences is crucial for the success of an architect in an international setting. It helps us understand these differences and, in doing so, improves our ability to react to specific behaviors that might have once seemed strange.



Turn the Ship Around! by L. David Marquet: Around reveals how one United States Navy captain managed to turn a dissatisfied submarine crew into a formidable and respected team. By changing how we think about leadership, this story will show you that we all have the power to be leaders inside.



Leadership is Language by L. David Marquet is a playbook for successful team management. It teaches leaders to change their language and mindsets to improve decision-making, empower workers, and achieve better results.



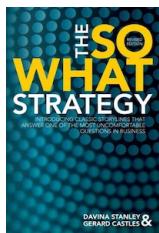
Nudge by Richard Thaler draws on psychology and behavioral economics research to define libertarian paternalism as an active engineering of choice architecture. The book also popularised the concept of a nudge, a choice architecture that predictably alters people's behavior without restricting options or significantly changing their economic incentives.



No Rules Rules by Erin Meyer and Reed Hastings provides insights into Netflix culture. When you give low-level employees access to information generally reserved for high-level executives, they get more done independently. They work faster without stopping to ask for information and approval. They make better decisions without needing input from the top.



Dare To Lead by Brené Brown defines a leader as anyone who takes responsibility for finding the potential in people and processes and dares to develop that potential.



The So What Strategy by Davina Stanley and Gerard Castles. Any architect has been caught at the end of a presentation when their audience, perhaps a leadership team or a Steering Committee, looks at them blankly and asks this most uncomfortable question: 'So what?' How does that help? The book provides a pragmatic approach to answering that question in one powerful sentence. Or how to set yourself up so nobody asks it.



Never Split the Difference by Chris Voss. Negotiations occur in many fields, such as business, and in some critical situations, like hostage situations. The book is a guide on how to behave best when certain things happen, regardless of whether that includes the need for negotiation techniques in hostage situations or in business (and architecture).

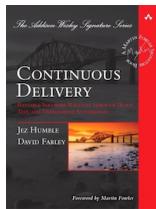
27.5: Organization and Processes



Accelerate — acceleration is the second derivative of position (speed being the first), so if you want to move faster, you need to accelerate. Sometimes, you need to jerk the system a bit, which is the proper term for the third derivative.

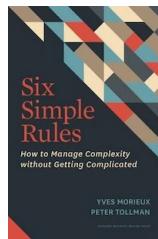


The Phoenix Project and **The Unicorn Project** by Gene Kim
The Phoenix Project is a best-selling novel about DevOps. The book's characters reveal through their actions why it's so important for organizations to put security first and tear down the silos that have traditionally existed between development and operations teams.

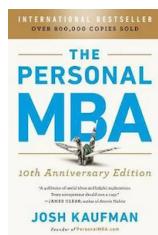


Continuous Delivery – through automation of the build, deployment, and testing process, and improved collaboration between developers, testers, and operations, delivery teams can get changes released in a matter of hours-sometimes even minutes-no matter what the size of a project or the complexity of its code base.

27.6: Business, Product, Strategy



Six Simple Rules: How to Manage Complexity without Getting Complicated by Yves Morieux and Peter Tollman: the book emphasizes that in today's complicated business environment, you must set up organizational structures based on cooperation. To deal with complexity, organizations should depend on the judgment of their people, which requires giving them more autonomy to act. It also depends on these people cooperating to utilize the organization's capabilities to cope with complex problems.



The Personal MBA 10th Anniversary Edition by Josh Kaufman provides an overview of the essentials of every major business topic: entrepreneurship, product development, marketing, sales, negotiation, accounting, finance, productivity, communication, psychology, leadership, systems design, analysis, and operations management.



Technology Strategy Patterns by Eben Hewitt provides a shared language—in the form of repeatable, practical patterns and templates—to produce great technology strategies.



Escaping the Build Trap by Melissa Perri is a practical guide to product management, product strategy, and ensuring product success.

28: Tools

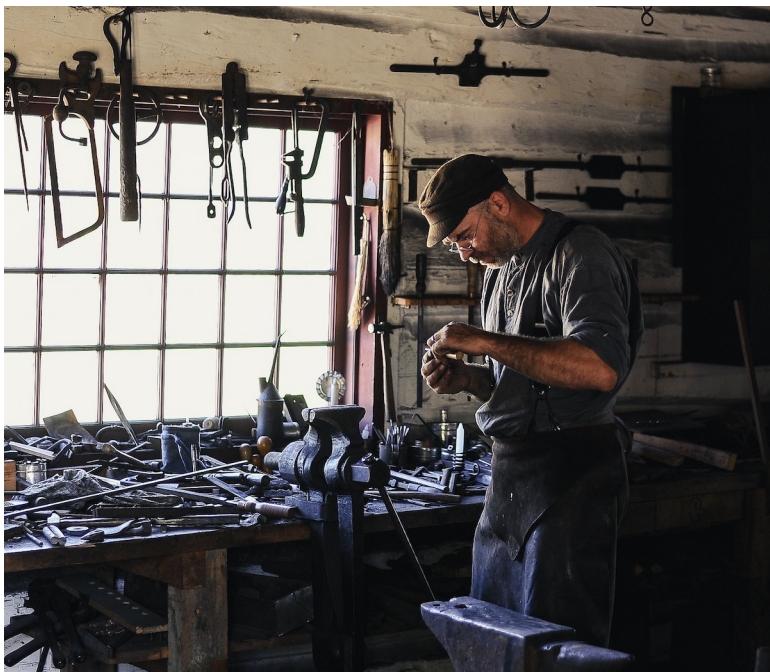


image by pexels from pixabay



Architecture Dashboard Examples: [source code of examples¹](#) of how to build an Architecture Data Dashboard as a part of the Grounded Architecture Data Foundation. The dashboard is a simple static website generated from JSON files and [published via GitHub pages²](#).



Sokrates³: Sokrates is a tool I built to implement my vision of documenting and analyzing software architectures of complex systems. Sokrates provides a pragmatic, inexpensive way to extract rich data from source code repositories. Sokrates can help you understand your code by making visible the size, complexity, and coupling of software and all people interactions and team topologies.

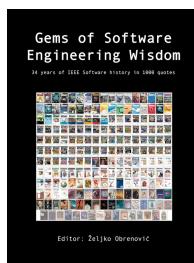
¹<https://github.com/zeljkoobrenovic/grounded-architecture-dashboard-examples>

²<https://zeljkoobrenovic.github.io/grounded-architecture-dashboard-examples/>

³<https://sokrates.dev>



Productivity Tools⁴: a collection of more than 100 online tools I built to help me in my daily work as an architect. I reuse these tools and lessons learned in building these tools when designing Data Foundation parts of the Grounded Architecture.



obren359.com⁵: I've created a curated collections and high-quality IT knowledge resources (articles, videos, podcasts).

⁴<https://obren.io/tools>

⁵<https://obren359.com>

29: Favorite Quotes



image by amy from pixabay

29.1: People, People, People

1

*Architecture is **not so much about the software, but about the people** who write the software. The core principles of architecture, such as **coupling and cohesion**, aren't about the code. The code doesn't 'care' about how cohesive or decoupled it is; ... But people do care about their coupling to other team members.* -James O. Coplien

*Software design is **an exercise in human relationships**.* -Kent Beck

*Bad software architecture is a people problem. When **people don't work well together they make bad decisions**.* -Kate Matsudaira

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

29.2: Changing Role of Architecture

1

*The role of architects has changed from trying to be the smartest person to **making everyone else smarter**.* -Gregor Hohpe

*The architect's role is changing from being primarily a decision maker to being a coordinator, advisor, and knowledge manager ... **a central knowledge hub.*** -Rainer Weinreich & Iris Groher

*Rather than inject itself in every decision loop, **architecture should design mechanisms for teams to make better decisions** - they'll know better anyhow, and slowing them down carries a high cost..* -Gregor Hohpe

*Your architecture team's job is to **solve your biggest problems**. The best setup is the one that allows it to accomplish that.* -Gregor Hohpe

*Your organization has to **earn its way to an effective architecture function**. You can't just plug some architects into the current mess and expect it to solve all your problems.* -Gregor Hophe

29.3: Complexity

1

Excessive complexity is nature's punishment for organizations that are unable to make decisions. -Gregor Hohpe

Metawork is more interesting than work. ... we love complicated little puzzles to solve, so we keep overengineering everything. -Neal Ford

Developers are drawn to complexity like moths to a flame often with the same result. -Neal Ford

29.4: Other

1

*Architecture is just a collective hunch, a **shared hallucination**, an assertion by a set of stakeholders on the nature of their observable world, be it a world that is or a world as they wish it to be.* -Grady Booch

***Architects code**, but not to deliver code. Rather, to understand the ramifications of the decisions that they're making.* -Gregor Hohpe, Dave Farley

*Architectural decisions are design decisions that are **hard to make** or **costly to change**.* -Olaf Zimmermann

Part VI: Appendix

30: Appendix Overview



image by pexels from pixabay

IN THIS SECTION, YOU WILL: Get an overview of three resources in the appendix.

As we bring this book to a close, I would like to equip you with a helpful appendix. In this part, I outline three resources that have proven vital in my journey as an IT architect and have reminded me about what I always need to know about crucial aspects of IT practice. I routinely refer back to them and carry them in my practitioner “backpack,” so I added their summaries in the appendix.

- **EIC/ISO 25010 Standard** focuses on product quality and system quality models. While imperfect, this standard is a reasonably complete yet compact source for understanding software maintainability, security, reliability, and performance efficiency.
- **Cloud Design Patterns** offer a mix of crucial distributed system and messaging system topics combined with modern public cloud engineering themes.
- **Characteristics of High Performing Organizations** from the ‘Accelerate’ book by Nicole Forsgren, Jez Humble, and Gene Kim is an excellent source of empirical knowledge about crucial practices of high-performing technology organizations.

31: ISO 25010 Standard



image by pexels from pixabay

IN THIS SECTION, YOU WILL: Get an overview of ISO/IEC 25010 standard that provides guidelines and recommendations for evaluating software product quality.

31.1: Overview

ISO/IEC 25010 is a standard that provides guidelines and recommendations for evaluating software product quality. It is a part of the ISO/IEC 25000 series, which encompasses a set of international standards for software engineering.

The standard identifies several quality characteristics that should be considered during evaluation. I have used four of them in practice and can recommend them as a pragmatic way to assess and discuss the quality of software systems:

- **Maintainability:** The ease with which the software can be modified, corrected, adapted, or enhanced.
- **Security:** The software's ability to protect information and data from unauthorized access, disclosure, alteration, destruction, or disruption.
- **Performance Efficiency:** The software's ability to perform its functions efficiently, considering resource utilization.
- **Reliability:** The software's ability to maintain a specified level of performance under stated conditions for a defined period.

31.2: Maintainability

Definition: The degree of effectiveness and efficiency with which the intended maintainers can modify a product or system.

31.2.1: Maintainability Characteristics

- **Analyzability:** The degree of effectiveness and efficiency with which it is possible to assess the impact on a product or system of an intended change to one or more of its parts or diagnose a product for deficiencies or causes of failures, or identify parts to be modified.
- **Modifiability:** The degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality.
- **Testability:** The degree of effectiveness and efficiency with which test criteria can be established for a system, product, or component, and tests can be performed to determine whether those criteria have been met.
- **Modularity:** The degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.
- **Reusability:** The degree to which an asset can be used in multiple systems or building other assets.

31.2.2: Maintainability Tactics

- **Keep Systems Small:** The overall size of the source code of the software product matters. Systems with about 200,000 lines of code are frequently challenging to maintain.
- **Organize System in Limited Number (e.g., 7±2) Balanced Components:** When a system has too many or too few

components, or if components differ in size significantly (e.g., you have 80% of code in one common component), it is considered more challenging to understand and maintain.

- **Couple Files Loosely:** Software products where more source code resides in files strongly coupled with other files are deemed more challenging to maintain.
- **Keep Components Loosely Coupled:** Hide most files in a component from direct dependencies from other components.
- **Avoid Duplication:** Avoid the occurrence of identical fragments of source code in more than one place in the product.
- **Write Short Units of Code:** Keep units (e.g., functions, methods) small, with many practitioners recommending units to be shorter than 30 lines. Short units are also much easier to unit test.
- **Write Simple Units of Code:** Keep the number of decision points (so-called McCabe index, e.g., if, while statement) per unit low, ideally below 5.
- **Avoid Many Parameters in Unit Interfaces:** A large number of parameters in a unit are deemed more complicated to maintain.

31.3: Security

The degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization.

31.3.1: Security Characteristics

- **Confidentiality:** Ensures that data are accessible only to those authorized.
- **Integrity:** Prevents unauthorized access or modifications.
- **Non-repudiation:** actions or events can be proven to have occurred.
- **Accountability:** Actions of an entity can be traced uniquely to the entity.
- **Authenticity:** The identity of a subject or resource can be proved to be the one claimed.

31.3.2: Security Tactics

- **Protect Data Transport:** Protect data transport with a sufficiently robust protection method, minimizing caching of sensitive data.
- **Protect Stored Data:** Prevent or restrict access to data stored physically. Encrypt sensitive data. Correctly applying a one-way hash to the data.
- **Verify Input and Output:** Reject invalid system input. The rejection should not disclose information. Validate within the system, validated against a whitelist. Prevent injection and overflow vulnerabilities. Escape all output. Never unnecessarily expose implementation details.

- **Uniquely Identify Actors:** Identify and record system actors in a way that points uniquely to a specific actor. Include detailed traceable information, such as location or origin.
- **Enforce In-Depth Authentication:** Enforce authentication for all system functions and all uses. Use an intrinsically robust authentication method. For failed authentication, do not perform any tasks or expose information.
- **Enforce In-Depth Authorization:** Authorize within the system so the user cannot circumvent it. Authorize for every system function and at every attempt. If authorization fails, record this event and inform the user only that authorization failed. Give users the least possible privileges.
- **Manage User Sessions Securely:** Create and expire sessions and tokens securely.
- **Keep Evidence:** Allow non-repudiation and accountability. Keep the proof that an actor actively approved and performed an action. Store it securely, and facilitate retrieval and analysis.
- **Manage Users Securely:** Implement a secure and automated process for user sign-up, blocking and removal, and management of user credentials.

31.4: Performance Efficiency

Definition: Performance relative to the resources used under stated conditions.

31.4.1: Performance Efficiency

- **Time Behavior:** The degree to which a product or system's response, processing times, and throughput rates meet requirements when performing its functions.
- **Capacity:** The degree to which the maximum limits of a product or system parameter meet requirements.
- **Resource Utilization:** The degree to which the amounts and types of resources a product or system use meet requirements when performing its functions.

31.4.2: Performance Efficiency Tactics

- **Observe System Performance:** Know the system's actual performance and support problem analysis and resolution by measuring and monitoring the system.
- **Optimize Internal Communication:** Limit the number of steps, usage of small messages, and transformations in inter-process communications.
- **Limit External Communication:** Limit usage of external services and associated uncertainty, especially if the system has no strong guarantees over the external service's time behavior.
- **Optimize Common Transactions:** Identify the most common and critical transactions, and apply standard strategies for their optimization.

- **Scale Transaction Capabilities:** Make it easy to increase transaction processing capacities when needed, both for individual components and the whole system.
- **Scale Data Capabilities:** Take care of the volume and characteristics of the data.
- **Isolate External Influences:** Control or exclude external influences that may impact the performance and the consistency of response times.
- **Provision Resources Elastically:** Accommodate variations in workload. so that the consumed computer resources and associated costs of a service rise and fall proportionally to the workload.

31.5: Reliability

The degree to which a system, product, or component performs specified functions under specified conditions for a specified period.

31.5.1: Reliability Dimensions

- **Maturity:** The system is thoroughly tested and has a low manual maintenance effort, minimizing the number of potential errors in production.
- **Availability:** The system is thoroughly tested and has a low manual maintenance effort, minimizing the number of potential errors in production.
- **Fault-Tolerance:** Fitted with mechanisms to ensure a certain error tolerance level, ensuring that not every error results in a system failure.
- **Recoverability:** Should the system fail despite all efforts, it has mechanisms to either recover fully automatically or support human intervention for fast recovery.

31.5.2: Reliability Tactics

- **Isolate Faults:** Prevent faults propagating from one to other components.
- **Prevent Inconsistent States (Transaction Handling):** Prevents inconsistent states and data in the presence of errors.
- **Avoid Single Points of Failure (Redundancy):** Redundancy determines to which extent a system is vulnerable to a single point of failure.
- **Automate Deployment:** The faster a system can be (re)deployed, the faster new versions can be put into

production, enabling more rapid recovery from errors and failures.

- **Make System Autonomous:** Avoid the dependency of a system on human intervention to stay operational.
- **Test Reliability:** Make test loads resemble production loads.
- **Implement Failover:** Make switching to another component easy when one fails.

32: Cloud Design Patterns



image by donna kirby from pixabay

IN THIS SECTION, YOU WILL: Get a mix of key distributed and messaging system topics combined with modern public cloud engineering themes.

32.1: Overview

Cloud Design Patterns offer a mix of key distributed and messaging system topics and modern public cloud engineering themes.

I grouped these patterns into the following categories:

- Performance and Scalability,
- Resiliency,
- Messaging,
- Management and Monitoring,
- Security, and
- Other.

Source: learn.microsoft.com/en-us/azure/architecture/patterns¹

¹<https://learn.microsoft.com/en-us/azure/architecture/patterns/>

32.2: Performance and Scalability

Make your system perform well under load and react well to the load change.

- **Command and Query Responsibility Segregation (CQRS):** Segregate operations that read data from operations that update data by using separate interfaces.
- **Event Sourcing:** Use an append-only store to record the entire series of events that describe actions taken on data in a domain.
- **Materialized View:** Generate prepopulated views over the data in one or more data stores when the data is not ideally formatted for required query operations.
- **Index Table:** Create indexes over the fields in data stores frequently referenced by queries.
- **Sharding:** Divide a data store into horizontal partitions or shards.
- **Static Content Hosting:** Deploy static content to a cloud-based storage service that can deliver them directly to the client.
- **Cache-Aside:** Load data on demand into a cache from a data store.
- **Throttling:** Control the consumption of resources used by an instance of an application, an individual tenant, or an entire service.
- **Rate Limit Pattern:** Limiting pattern to help you avoid or minimize throttling errors related to these throttling limits and to help you more accurately predict throughput.
- **Geodes:** Deploy a collection of backend services into geographical nodes, each of which can service any request for any client in any region.

32.3: Resiliency

Gracefully handle and recover from failures.

- **Bulkhead:** Isolate elements of an application into pools so that if one fails, the others will continue to function.
- **Retry:** Enable an application to handle anticipated, temporary failures when connecting to a service or network resource by transparently retrying a previously failed operation.
- **Circuit Breaker:** Handle faults that might take a variable amount of time to fix when connecting to a remote service or resource.
- **Compensating Transaction:** Undo the work performed by a series of steps, which together define an eventually consistent operation.
- **Leader Election:** Coordinate the actions performed by a collection of collaborating task instances in a distributed application by electing one instance as the leader responsible for managing the other instances.

32.4: Messaging

Create a messaging infrastructure that connects the components and services, ideally loosely coupled, to maximize scalability.

- **Publisher/Subscriber:** Enable an application to announce events to multiple interested consumers asynchronously without coupling the senders to the receivers.
- **Competing Consumers:** Enable multiple concurrent consumers to process messages received on the same messaging channel.
- **Pipes and Filters:** Break down a task that performs complex processing into a series of separate elements that can be reused.
- **Priority Queue:** Prioritize requests sent to services so that requests with a higher priority are received and processed more quickly than those with a lower priority.
- **Queue-Based Load Leveling:** Use a queue that acts as a buffer between a task and a service that it invokes to smooth intermittent heavy loads.
- **Scheduler Agent Supervisor:** Coordinate actions across a distributed set of services and other remote resources.
- **Asynchronous Request-Reply:** Decouple backend processing from a frontend host, where backend processing needs to be asynchronous, but the frontend still needs a clear response.
- **Choreography:** Have each system component participate in the decision-making process about the workflow of a business transaction instead of relying on a central point of control.
- **Saga:** Manage data consistency across microservices in distributed transaction scenarios. A saga is a sequence of transactions that updates each service and publishes a message or event to trigger the next transaction step.
- **Sequential Convoy:** Process a set of related messages in a defined order without blocking the processing of other groups of messages.

32.5: Management and Monitoring

Expose runtime information that administrators and operators can use to manage and monitor the system—supporting changing business requirements and customization without requiring the application to be stopped or redeployed.

- **Health Endpoint Monitoring:** Implement functional checks in an application that external tools can access through exposed endpoints at regular intervals.
- **Ambassador:** Create helper services that send network requests on behalf of a consumer service or application.
- **Anti-Corruption Layer:** Implement a façade or adapter layer between a modern application and a legacy system.
- **External Configuration Store:** Move configuration information from the application deployment package to a centralized location.
- **Gateway Aggregation:** Use a gateway to aggregate multiple individual requests into a single request.
- **Gateway Offloading:** Offload shared or specialized service functionality to a gateway proxy.
- **Gateway Routing:** Route requests to multiple services using a single endpoint.
- **Sidecar:** Deploy components of an application into a separate process or container to provide isolation and encapsulation.
- **Strangler Fig:** Incrementally migrate a legacy system by gradually replacing specific pieces of functionality with new applications and services.

32.6: Security

Prevent malicious or accidental actions outside of the designed usage, and prevent disclosure or loss of information.

- **Federated Identity:** Delegate authentication to an external identity provider.
- **Gatekeeper:** Protect applications and services using a dedicated host instance that is a broker between clients and the application or service, validates and sanitizes requests, and passes requests and data between them.
- **Valet Key:** Use a token or key that provides clients restricted direct access to a specific resource or service.
- **Claim Check:** Split a large message into a claim check and a payload.

32.7: Other Patterns

Create good designs. Take care of consistency, coherence, maintainability, and reusability.

- **Compute Resource Consolidation:** Consolidate multiple tasks or operations into a single computational unit.
- **Backends for Frontends:** Create different backend services to be consumed by specific frontend applications or interfaces.
- **Deployment Stamps:** Deploy multiple independent copies of application components, including data stores.

33: High Performing Technology Organizations

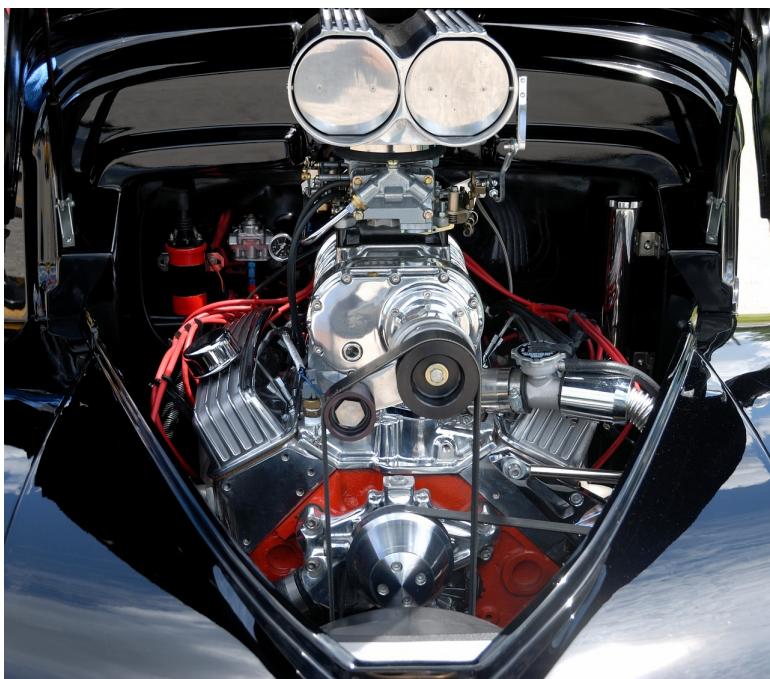


image by paul brennan from pixabay

IN THIS SECTION, YOU WILL: Get a summary of the characteristics of high-performing technology organizations.

33.1: Overview

Characteristics of High Performing Organizations from the [Accelerate book](#)¹ by Nicole Forsgren, Jez Humble, and Gene Kim is an excellent source of empirical knowledge about high-performing IT organizations.

I summarized several critical insights from this book:

- Overview of **four key metrics**, describing metrics that provide suitable measures of organization performance.
- Overview of **critical practices of high-performing technology organizations** grouped into the following categories: Continuous Delivery, Architecture, Product and Process, Lean Management and Monitoring Capabilities, Culture

¹<https://www.oreilly.com/library/view/accelerate/9781457191435/>

33.2: Four Key Metrics

- **Lead time for changes:** Elite performers have a lead time for changes of less than 1 hour (from code committed to code successfully running in production).
- **Deployment frequency:** Elite performers have a deployment frequency of multiple times daily.
- **Time to restore service:** Elite performers have a mean time to recover (MTTR) of less than 1 hour.
- **Change failure rate:** Elite performers have a 0-15% change failure rate.

33.3: Practices

33.3.1: Continuous Delivery

- **Use version control for all production artifacts:** For all production artifacts, including application code, application configurations, system configurations, and scripts for automating the build and configuration of the environment.
- **Automate your deployment process:** The degree to which deployments are fully automated and do not require manual intervention.
- **Implement continuous integration:** Code is regularly checked in, and each check-in triggers a set of quick tests to discover serious regressions, which developers fix immediately.
- **Use trunk-based development methods:** Fewer than three active branches; branches and forks having very short lifetimes (e.g., less than a day); teams rarely or never having “code lock” periods.
- **Implement test automation:** Software tests are run automatically (not manually) continuously through the development process.
- **Support test data management:** Test data requires careful maintenance, and test data management is becoming an increasingly important part of automated testing.
- **Shift left on security:** Integrating security into the design and testing phases of the software development process is vital to driving IT performance.
- **Implement continuous delivery (CD):** Software is in a deployable state throughout its lifecycle, and the team prioritizes keeping the software in a deployable state over working on new features.

33.3.2: Architecture

- **Use a loosely coupled architecture:** The extent to which a team can test and deploy their applications on demand without requiring orchestration with other services.
- **Architect for empowered teams:** Teams that can choose which tools to use, do better, and, in turn, drive better software development and delivery performance.

33.3.3: Product and Process

- **Gather and implement customer feedback:** Actively and regularly seek customer feedback and incorporate this feedback into the design of products.
- **Make the flow of work visible through the value stream:** Teams should have a good understanding of and visibility into the flow from the business to customers, including the status of products and features.
- **Working in small batches:** Teams should slice work into small pieces that can be completed in a week or less.
- **Foster and enable team experimentation:** Team experimentation is the ability of developers to try out new ideas and create and update specifications during the development process without requiring approval from outside of the team, which allows them to innovate quickly and create value.

33.3.4: Lean Management and Monitoring Capabilities

- **Have a lightweight change approval process:** A lightweight change approval process based on peer review (pair programming or intrateam code review) produces

superior IT performance than using external change approval boards (CABs).

- **Monitor across applications and infrastructure to inform business decisions:** Use data from application and infrastructure monitoring tools to take action and make business decisions. This monitoring goes beyond paging people when things go wrong.
- **Check system health proactively:** Monitor system health using threshold and rate-of-change warnings to enable teams to detect and mitigate problems preemptively.
- **Improve process and manage work with work-in-progress (WIP) limits:** Using work-in-progress limits to manage work flow is well known in the Lean community. When used effectively, this drives process improvement, increases throughput and makes constraints visible in the system.
- **Visualize work to monitor quality and communicate throughout the team:** Visual displays, such as dashboards or internal websites, used to monitor quality and work in progress have contributed to software delivery performance.

33.3.5: Culture

- **Support a generative culture (as outlined by Westrum):** Hallmarks of this measure include good information sharing, high cooperation and trust, bridging between teams, and conscious inquiry.
- **Encourage and support learning:** Is learning, in your culture, considered essential for continued progress? Is learning thought of as a cost or an investment?
- **Support and facilitate collaboration among teams:** Reflects how well traditionally siloed teams interact in development, operations, and information security.

- **Provide resources and tools that make work meaningful:** This measure of job satisfaction is about doing challenging and meaningful work and being empowered to exercise your skills and judgment.
- **Support or embody transformational leadership:** Comprised of five factors: vision, intellectual stimulation, inspirational communication, supportive leadership, and personal recognition.