Grounded Architecture

Part III: Expanding Horizons

Željko Obrenović

Grounded Architecture

Redefining IT Architecture Practice in the Digital Enterprise

Željko Obrenović

This book is available at https://leanpub.com/groundedarchitecture

This version was published on 2025-05-26

Contents

Part I: On Human Complexity

1: On Human Complexity: Introduction

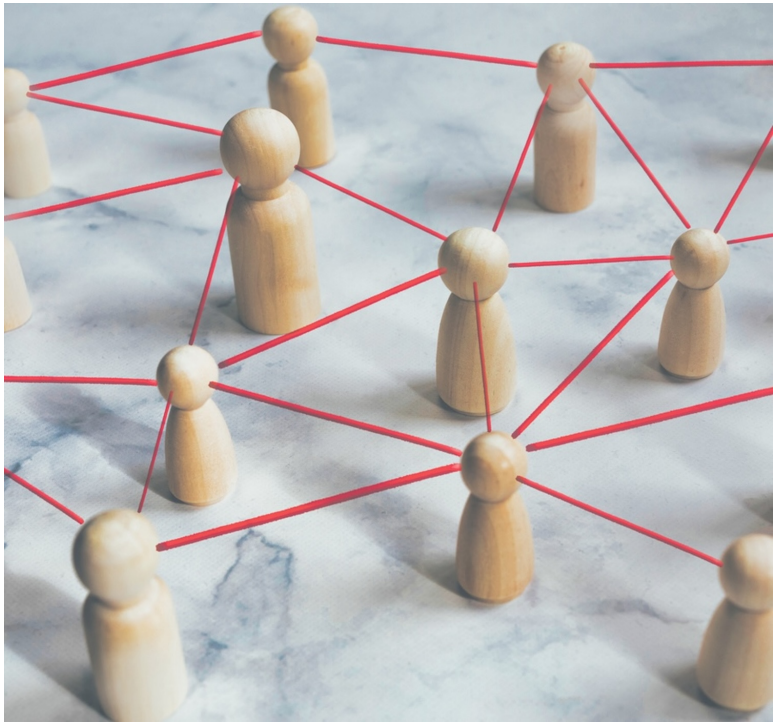

image by cerro photography from istock

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

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

The following sections will explore:

- **The Culture Map: Architects' Culture Mindfield Compass**: Navigating cultural diversity is crucial in multinational organizations. Erin Meyer's "The Culture Map" offers invaluable guidance for IT architects to collaborate seamlessly with colleagues from different cultural backgrounds.
- **Cooperation-Based Organizations: Six Simple Rules**: Grounded Architecture is particularly well-suited for organizations organized based on cooperation. As businesses grow increasingly complex, the importance of cooperation within organizational structures cannot be overstated. The "Six Simple Rules" framework emphasizes that successful organizations leverage their members' collective judgment and cooperation. By fostering a culture of collaboration, organizations can better utilize their capabilities to address and resolve complex problems.
- **The Human Side of Decision-Making**: Decision-making is not purely a logical process; it is deeply influenced by human psychology. Architects must be aware of cognitive biases that can skew their judgments. Outcome bias, hindsight bias, and confirmation bias are just a few of the common pitfalls that can affect decision-making. By recognizing these biases, architects can implement strategies to mitigate their effects, leading to more balanced and objective decisions. This understanding is crucial for fostering a culture of rational and evidence-based decision-making within IT teams.
- **Effortless Architecture**: Complexity is a pervasive challenge in the tech industry, often leading to inefficiencies and unsustainable projects. Inspired by Greg McKeown's book "Effortless," this section emphasizes the importance of simplifying tasks and processes. Architects can create more streamlined and effective architectural solutions by focusing on what truly matters and eliminating unnecessary complexity. McKeown's principles encourage architects to adopt a mindset of ease and efficiency, which can lead to more sustainable and manageable projects. Architects can enhance

productivity and foster a more innovative environment by reducing the cognitive load and focusing on essential activities.

2: The Culture Map: Architects' Culture Compass

image by maik from pixabay

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

KEY POINTS:

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

In multinational organizations, architects will need to work with many different cultures. **Awareness of cultural differences** is even more important for architects, as they are bridging diverse cultures and domains (technology, business, product, organization).

The work of Erin Meyer, **The Culture Map**[1], is a beneficial tool for working harmoniously with people from various cultures and backgrounds. Meyer's model contains **eight scales**, each representing a key area, showing how cultures vary from extreme to extreme. The eight scales describe a continuum between the two ends which are diametric opposite or competing positions:

- **Communicating** – Are cultures low-context (simple, verbose, and clear) or high-context (rich deep meaning in interactions)?
- **Evaluating** – When giving negative feedback, does one give it directly or prefer being indirect and discreet?
- **Persuading** – Do people like to hear specific cases and examples or prefer detailed holistic explanations?
- **Leading** – Are people in groups egalitarian or prefer hierarchy?
- **Deciding** – Are decisions made in consensus or made top-down?
- **Trusting** – Do people base trust on how well they know each other or how well they work together?
- **Disagreeing** – Are disagreements tackled directly, or do people prefer to avoid confrontations?

[1] https://erinmeyer.com/books/the-culture-map/

- **Scheduling** – Do people see time as absolute linear points or consider it a flexible range?

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

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

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

2.1: Communicating

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

image by luckybusiness from istock

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

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

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

[2] https://architectelevator.com/strategy/complex-topics-stick/

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

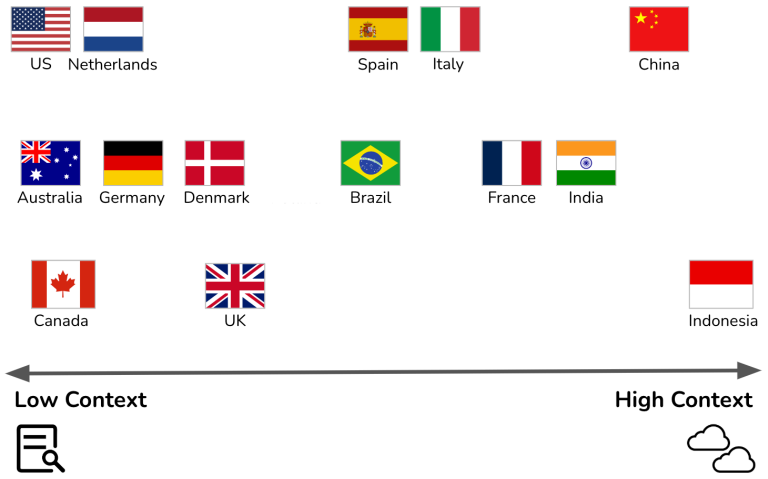

***Figure 1**: The Communicating scale: a graphical representation of some Culture Map countries (the cultural context of my recent professional interactions).*

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

- **Misunderstanding Due to High-Context Communication**:
  - **Scenario**: An architect from a high-context culture presents a new system design to a global team. The architect uses indirect language, implying certain requirements or risks without

stating them explicitly, expecting the team to understand the subtleties and read between the lines.
  - **Problem**: Team members from low-context cultures may miss critical nuances or interpret the communication literally, leading to misunderstandings about the scope, priorities, or potential risks in the project. This issue can result in errors in implementation, overlooked risks, or misaligned expectations.

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

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

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

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

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

  - **Scenario**: During a project review, an architect from a low-context culture directly criticizes a design choice made by a team member from a high-context culture, expecting the feedback to be taken at face value and used for improvement.

  - **Problem**: The team member might perceive the direct criticism as rude or disrespectful, leading to tension or a breakdown in collaboration. In high-context cultures, people often give feedback subtly to maintain harmony, so the direct approach might cause more harm than good.

- **Confusion Over Ambiguous Requirements**:

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

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

2.2: Evaluating

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

image by rickey123 from pixabay

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

Providing constructive **criticism in the right way** is crucial for architects to make any impact. Sometimes the same feedback will lead to different reactions, even within the same teams with members from diverse backgrounds. Being too positive in some cultures leads to underestimation of the significance of the feedback. Being too negative may result in pushback and rejection. In my experience, architects need to adapt their

feedback to the audience and do lots of "**duplication" by presenting the same feedback differently** to diverse groups.

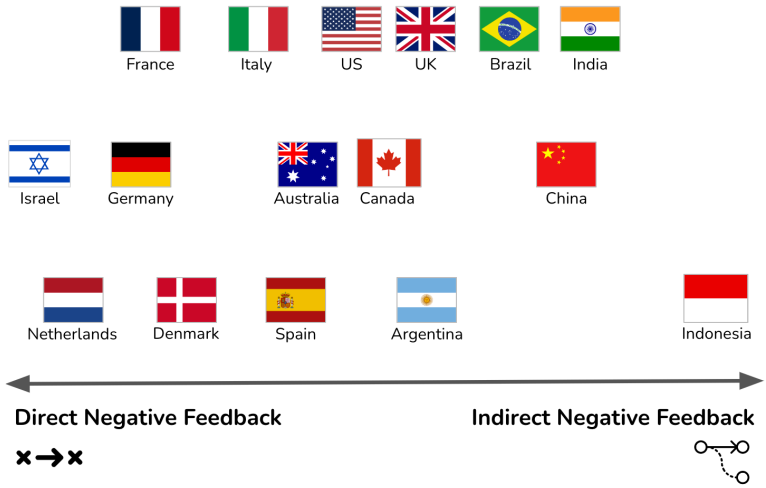

Figure 2: *The Evaluating scale: a graphical representation of some Culture Map countries (the cultural context of my recent professional interactions).*

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

- **Overly Direct Feedback Leading to Resistance**:
  - **Scenario**: An architect from a culture that values directness in feedback reviews a design document created by a team member from a culture that values more diplomatic feedback. The architect bluntly points out flaws in the design, focusing on what is wrong without much positive reinforcement.
  - **Problem**: The team member from the diplomatic culture may perceive the feedback as overly harsh or even disrespectful. Instead of motivating improvement, this direct approach may

lead to defensiveness, loss of face, or reluctance to engage in future discussions. Though technically valid, the criticism may not result in the desired improvements due to cultural misalignment.

- **Overly Diplomatic Feedback Leading to Misunderstanding**:
  - **Scenario**: An architect from a culture that values indirect communication provides feedback on a technical proposal from a team member from a culture that expects more straightforward feedback. The architect couches the criticism positively, saying, "This is a great start, but maybe we could consider some adjustments."
  - **Problem**: Team members from the direct-feedback culture may not grasp the seriousness of the issues being raised. The subtlety might lead them to underestimate the need for changes, resulting in a final product that falls short of expectations. The feedback is not acted upon as strongly as it should be, leading to potential project setbacks.

- **Cultural Differences in Group Settings**:
  - **Scenario**: During a design review meeting, an architect from a culture that values group harmony provides criticism very indirectly, using vague terms to avoid singling out any individual. The meeting includes team members from cultures more accustomed to direct feedback in group settings.
  - **Problem**: The more direct cultures may find the feedback unhelpful or confusing, as the indirect criticism doesn't identify the issues or the individuals responsible for them. This problem could lead to inaction or misunderstandings about what the team must address. On the other hand, if the architect tries to adapt and provides more direct feedback, it might cause discomfort or embarrassment for those not used to such an approach.

- **Mixed Reactions in Diverse Teams**:
  - **Scenario**: An architect leads a multinational team with members from various cultural backgrounds, including

high-context and low-context cultures. During a code review, the architect provides criticism in a balanced way, aiming to be constructive by combining positive feedback with suggestions for improvement.
  - **Problem**: The feedback is received differently across the team. Team members from cultures that prefer direct feedback may feel that the criticism is too soft and doesn't address the real issues. In contrast, those from cultures that prefer more diplomatic feedback may feel the criticism is too harsh. This results in confusion and varying degrees of engagement with the feedback, making it difficult for the team to move forward cohesively.

- **Feedback in Written vs. Verbal Form**:
  - **Scenario**: An architect from a culture that values written feedback for clarity and record-keeping provides detailed written criticism of a system architecture proposal. The feedback is sent to a team member from a culture that prefers face-to-face communication for sensitive matters.
  - **Problem**: The written feedback, although clear and precise, may be perceived as impersonal or even confrontational by the team member, especially if it touches on significant flaws. The lack of a personal touch might lead to misinterpretation of the architect's tone and intent, reducing the effectiveness of the feedback and potentially harming the working relationship.

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

2.3: Persuading

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

image by fizkes from istock

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

Architects must be able to **persuade in both applications-first and principles-first ways**. In addition to cultural differences, the additional complication comes from talking to diverse audiences. For instance, C-

level executives typically have less time and may prefer applications-first presentations ("get to the point, stick to the point"). While in other parts of the company, you may need to spend a long time carefully building the argument following the principal first approach. I typically aim to prepare well for both, having a short management summary and easily retrievable all supporting evidence.

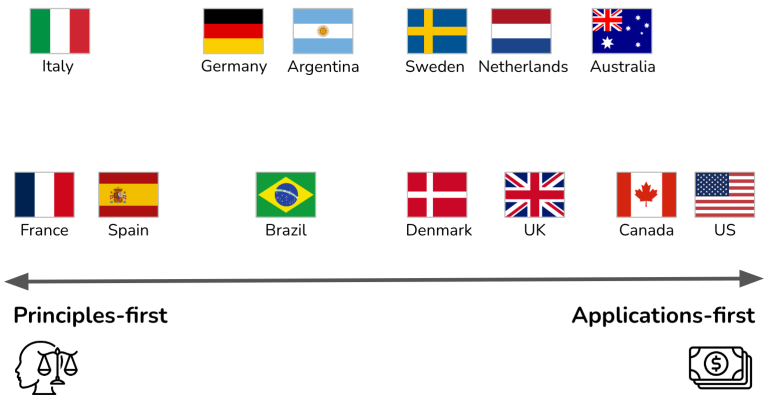

***Figure 3**: The Persuading scale: a graphical representation of some Culture Map countries (the cultural context of my recent professional interactions).*

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

- **Mismatch Between Holistic and Specific Thought Patterns**:
  - **Scenario**: An architect from a Western culture presents a detailed, step-by-step plan for a new software architecture to a team that includes members from East Asian cultures. The architect breaks down the argument into distinct components, expecting the team to follow the logic sequentially.

  - **Problem**: The team members from East Asian cultures, who may prefer holistic thinking, might struggle to see how these separate components fit into the broader system. They might find the argument less convincing because it doesn't address the overall harmony and integration of the system from the outset. As a result, the team meets the architect's proposal with skepticism or requests for a more integrated explanation.

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

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

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

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

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

  - **Scenario**: An architect works with a diverse project team that includes members from holistic and specific-thinking cultures. The architect tries to persuade the team to adopt a new project management tool by explaining its detailed features and benefits in a specific, component-by-component manner.
  - **Problem**: The holistic thinkers on the team might find this argument unconvincing because it doesn't address how the tool fits into the broader workflow or organizational goals. On the other hand, the specific thinkers might be satisfied with the details but may miss the overall strategic alignment. The architect's failure to balance both thought patterns results in partial buy-in, with some team members remaining unconvinced.

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

  - **Scenario**: An architect trained in a Western analytical tradition tries to persuade a multicultural team about the superiority of microservices architecture by focusing on empirical evidence and logical analysis. This approach is rooted in their educational background, emphasizing critical thinking and empirical validation.
  - **Problem**: Team members from cultures with a more conceptual approach to problem-solving might find this evidence-based approach lacking in conceptual depth. They may prefer a discussion that starts with overarching principles or theoretical considerations about system design before delving into the specifics. This mismatch can lead to difficulty in gaining complete consensus.

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

2.4: Leading

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

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

image by jacoblund from istock

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

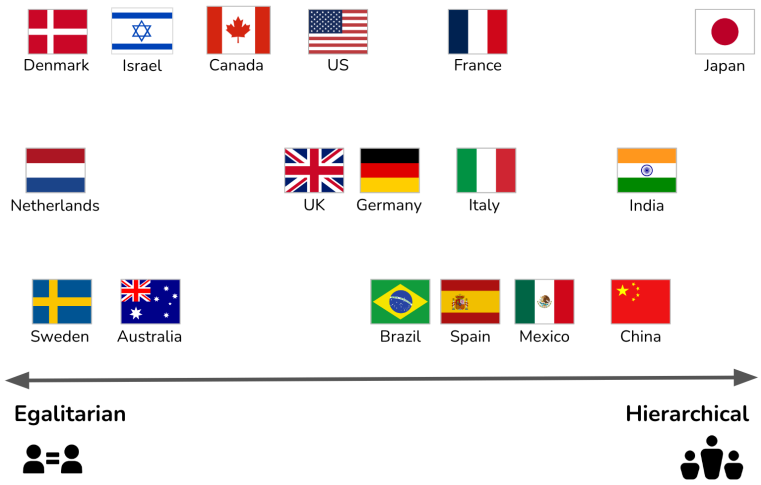

***Figure 4**: The Leading scale: a graphical representation of some Culture Map countries (the cultural context of my recent professional interactions).*

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

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

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

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

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

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

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

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

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

2.5: Deciding

Architectural work is about making decisions[3]. The Deciding scale measures the degree to which a culture is **consensus-minded**.

image by fizkes from istock

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

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

[3]https://architectelevator.com/gregors-law/

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.

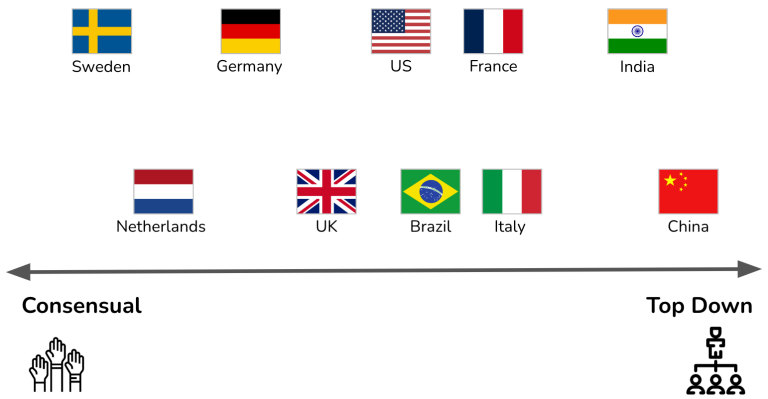

***Figure 5**: The Deciding scale: a graphical representation of some Culture Map countries (the cultural context of my recent professional interactions).*

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

- **Conflict Between Consensus and Hierarchical Decision-Making**:
  - **Scenario**: An architect from a hierarchical culture is leading a project team with members from both hierarchical and consensus-oriented cultures, including team members from different parts of Europe and the United States. The architect is used to making decisions after consulting with a few key stakeholders but expects to have the final say.
  - **Problem**: The European team members expect a more consensus-driven approach, where decisions are made only after thorough discussion and agreement from the entire group. Meanwhile, the American team members might

be more comfortable with the architect making a quick, top-down decision after considering input. The differing expectations lead to frustration: some members feel excluded when decisions are made too quickly, and others may see the process as unnecessarily slow and cumbersome. This results in delays and dissatisfaction within the team.

- **Delayed Decision-Making Due to Consensus Expectations**:
  - **Scenario**: An architect is working with a team, where decisions are typically made through a slow, consensus-driven process that involves input from all levels of the hierarchy. The architect, coming from a culture that values quicker decision-making, becomes frustrated with the time it takes to reach an agreement.
  - **Problem**: The architect's attempts to expedite the decision-making process are met with resistance, as the Japanese team members are uncomfortable deciding without complete consensus. This problem leads to delays in the project as the architect struggles to reconcile the need for timely decisions with the team's cultural expectation for consensus. The architect may inadvertently cause tension by pushing for a decision before the team is ready.
- **Decision-Making Deadlock in a Culturally Diverse Team**:
  - **Scenario**: An architect leads a multicultural team tasked with selecting a new technology stack. The team includes members from different parts of Europe (consensus-oriented, egalitarian), Asia (hierarchical, consensus-oriented), and the United States (individualistic, often leader-driven). The architect attempts to reach a consensus but finds discussions drag on without a clear decision.
  - **Problem**: Some team members are reluctant to finalize a decision without complete group agreement, while others expect the architect to step in and make a decisive call when discussions stall. The architect, unsure how to proceed, may either push through a decision, alienating some team members, or allow the discussions to continue indefinitely, frustrating others who expect a quicker resolution. This problem leads to a deadlock, with the project stalling due to the inability to make timely decisions.

- **Perception of Indecisiveness Due to Consensus-Building**:
  - **Scenario**: An architect from a consensus-driven culture is working with a team from a culture that values quick, top-down decision-making. The architect spends considerable time gathering input and seeking agreement from all stakeholders before deciding on a critical architectural change.
  - **Problem**: The team members perceive the architect's careful consensus-building approach as indecisiveness or lacking leadership. They may become frustrated with the slow pace and lose confidence in the architect's ability to lead the project effectively. This problem can result in a lack of cohesion, with team members potentially bypassing the architect to push for quicker decisions through other channels.

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

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

2.6: Trusting

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

image by scyther5 from istock

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

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

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

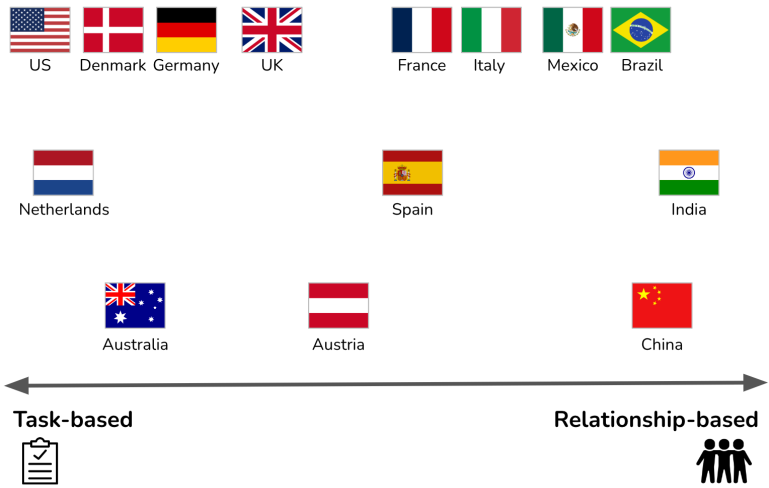

***Figure 6**: The Trusting scale: a graphical representation of some Culture Map countries (the cultural context of my recent professional interactions).*

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

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

- **Misalignment in Trust-Building Expectations with Clients**:

  - **Scenario**: An architect from a relationship-based culture works with a client from a task-based culture. The architect invests time in getting to know the client personally, sharing meals, and engaging in small talk, believing this will build a strong foundation of trust.
  - **Problem**: The client, who values task-based trust, may perceive these efforts as unnecessary or a waste of time. They might feel that the architect is not focused enough on delivering results and might become frustrated with what they see as a lack of professionalism. This problem can lead to a strained relationship, with the client doubting the architect's ability to deliver on technical promises.

- **Difficulty in Integrating into a Relationship-Based Team**:

  - **Scenario**: An architect from a task-based culture joins a team in a company that operates within a relationship-based culture. The architect is eager to start working on the project and skips social events, such as team lunches or informal gatherings, to focus on technical tasks.
  - **Problem**: The team may view the architect as aloof or uninterested in building a personal connection, which is crucial in their culture for establishing trust. As a result, the team might be less willing to collaborate openly, share information, or support the architect's initiatives. This problem leads to a lack of cohesion and potentially undermines the project's success.

- **Challenges in Gaining Stakeholder Trust Across Cultures**:

  - **Scenario**: An architect works with stakeholders from task- and relationship-based cultures. The architect prioritizes building trust with the task-based stakeholders by consistently delivering on project milestones while attempting to build trust with relationship-based stakeholders through regular social interactions and personal engagement.
  - **Problem**: The architect struggles to balance these approaches, potentially leading to dissatisfaction on both sides. Task-based stakeholders may feel that the architect is spending too much time on "soft" activities, while relationship-based

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

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

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

2.7: Disagreeing

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

image by fizkes from istock

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

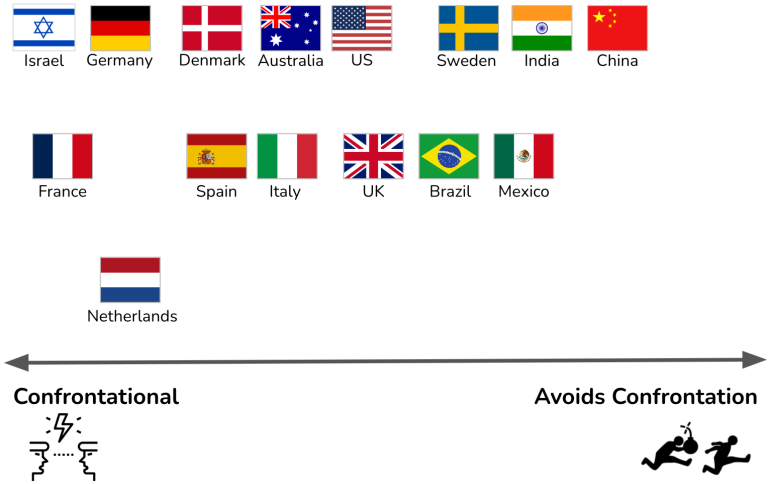

***Figure** 7: The Disagreeing scale: a graphical representation of some Culture Map countries (the cultural context of my recent professional interactions).*

In Software Engineering and IT Architecture, disagreements are inevitable, especially when dealing with complex projects and diverse teams. Here are some examples related to the "Disagreeing" dimension:

- **Perceived Aggressiveness in Direct Disagreement**:
  - **Scenario**: An architect from a culture that values direct communication and open debate is leading a discussion on a new system design with a team that includes members from cultures that avoid confrontation. The architect openly challenges ideas and expects others to do the same, believing this challenge will lead to the best outcome.
  - **Problem**: Team members from cultures that view open disagreement as disruptive may find the architect's direct approach aggressive or rude. They might withdraw from the discussion, avoid sharing their opinions, or feel uncomfortable contributing, which can stifle creativity and collaboration. This problem could lead to a lack of diverse input in the design process and potentially poorer decisions.
- **Avoidance of Critical Feedback**:

  - **Scenario**: An architect from a culture that prefers to avoid open conflict works with a team that sees disagreement as a healthy part of decision-making. When presented with a flawed design, the architect hesitates, openly criticizing the work and offering indirect feedback that the design "might need some improvement."
  - **Problem**: The German team members might interpret the architect's indirect feedback as approval or as a sign that the issues are minor. As a result, they may proceed with the flawed design, unaware of the architect's true concerns. This can lead to significant problems later in the project when the issues become more apparent, and more costly to fix. The architect's reluctance to engage in open disagreement may lead to misunderstandings and suboptimal outcomes.

- **Misinterpretation of Passionate Debate**:

  - **Scenario**: An architect from a culture that embraces passionate debate engages in a heated discussion with colleagues who value harmony and consensus. The architect raises their voice and uses strong language to emphasize their points, a sign of commitment and engagement.
  - **Problem**: The Malaysian team members may interpret the passionate debate as anger or personal conflict, which they might find unsettling. They may avoid further confrontation or seek to smooth over the disagreement without fully addressing the issues, leading to unresolved conflicts. The architect's approach, while intended to spark productive debate, might instead lead to discomfort and avoidance of future discussions.

- **Challenges in Reaching Consensus**:

  - **Scenario**: An architect from a culture that values consensus and group harmony is working with a team from a culture that is comfortable with open disagreement and individual opinions. During a meeting, the architect avoids voicing disagreement with a proposed solution, preferring to seek a behind-the-scenes resolution.

  - **Problem**: The American team members might perceive the lack of open disagreement as agreement or indifference, leading them to push forward with the solution without further discussion. Later, when issues arise, the architect's unvoiced concerns might surface, causing frustration among the team members who thought the decision had been fully supported. The differing approaches to disagreement can result in misalignment and project delays.

- **Conflict Avoidance Leading to Poor Decision-Making**:

  - **Scenario**: An architect is leading a multicultural team where some members are from cultures that highly value avoiding conflict and others from cultures that see conflict as a natural part of decision-making. The architect, aware of the cultural differences, tries to avoid open disagreements to keep the team harmonious.
  - **Problem**: Critical issues and differing opinions may not be fully explored by avoiding confrontation, leading to decisions that do not consider all perspectives. Some team members might feel that important debates are being stifled, leading to dissatisfaction and disengagement. Meanwhile, the team members who prefer to avoid conflict may feel uncomfortable if disagreements are forced. This problem can result in decisions that are not fully vetted, potentially leading to issues later in the project.

These examples illustrate how different cultural attitudes towards disagreement can lead to communication challenges in software engineering and IT architecture. Architects must navigate these differences carefully, finding ways to encourage constructive debate without alienating team members or stifling essential discussions. Open conversations about handling disagreements help align expectations and improve team dynamics.

2.8: Scheduling

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

image by bobex_73 from istock

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

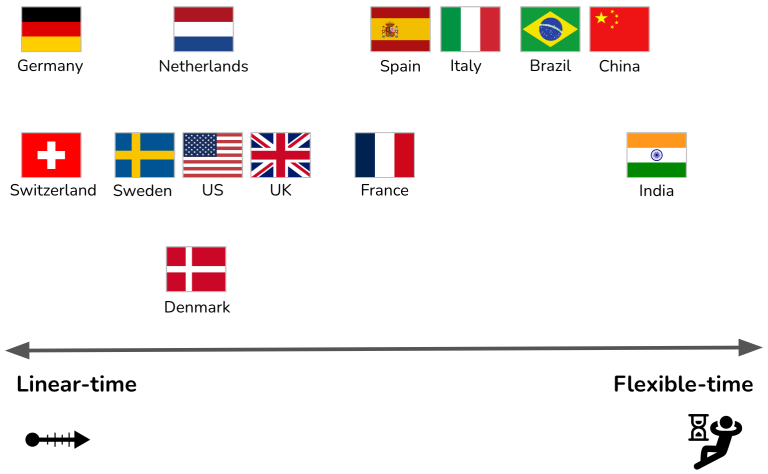

***Figure 8**: The Scheduling scale: a graphical representation of some Culture Map countries (the cultural context of my recent professional interactions).*

In IT architecture and Software Engineering, scheduling and time management are critical, but different cultural attitudes toward time can lead to communication problems. Here are some examples related to the "Scheduling" dimension:

- **Tension Due to Strict Adherence to Schedules**:
  - **Scenario**: An architect from a monochronic culture, where punctuality and strict adherence to schedules is highly valued, is leading a project with a team that includes members from a polychronic culture, where schedules are more flexible and interruptions are common.
  - **Problem**: The architect becomes frustrated when team members frequently arrive late to meetings or discussions veer off the planned agenda. The polychronic team members perceive the architect's insistence on sticking to the schedule as inflexible or overly rigid. This tension can create an uncomfortable working environment, with the architect feeling disrespected and the team feeling pressured and misunderstood.
- **Project Delays Due to Flexible Scheduling Attitudes**:

  - **Scenario**: An architect from a polychronic culture, where multitasking and adjusting plans based on the situation are everyday, is managing a project for a client from a monochronic culture. The architect is comfortable changing the project timeline as new tasks emerge or priorities shift.
  - **Problem**: The client, who expects a linear and predictable schedule, becomes concerned when deadlines are missed or the project timeline seems fluid. They perceive the architect as disorganized or unprofessional, leading to a loss of trust. The architect's flexible approach to scheduling conflicts with the client's expectations, resulting in dissatisfaction and potential conflicts over project management practices.

- **Misalignment in Meeting Expectations**:

  - **Scenario**: An architect in a multicultural team sets up regular meetings to review project progress. The team includes members from both monochronic cultures and polychronic cultures. The architect expects everyone to arrive on time, follow the agenda closely, and finish the meeting within the scheduled time.
  - **Problem**: Team members from the polychronic culture might arrive late, extend discussions beyond the agenda, or prioritize ongoing tasks over the meeting. This problem frustrates the monochronic team members who value punctuality and structure. The meeting becomes less productive, with some members feeling their time is not respected, while others feel rushed or constrained by the rigid schedule.

- **Challenges in Coordinating Cross-Cultural Project Teams**:

  - **Scenario**: An architect is coordinating a project involving teams from both monochronic and polychronic cultures. The project requires collaboration across multiple time zones, with some teams expecting strict adherence to the project timeline and others seeing the schedule as more of a guideline.
  - **Problem**: The architect struggles to keep the project on track because the teams have different expectations about deadlines and meeting times. The monochronic teams may become frustrated when deliverables are delayed due to the more flexible approach of the polychronic teams, leading to friction and

blame-shifting. The project risks falling behind schedule as the architect tries to balance these differing cultural attitudes toward time management.

- **Perception of Disrespect Due to Scheduling Flexibility**:
  - **Scenario**: An architect from a monochronic culture is working with a vendor from a polychronic culture to implement a new software system. The architect sends detailed schedules and expects the vendor to follow them closely. However, the vendor often changes meeting times or shifts deadlines based on other priorities.
  - **Problem**: The architect perceives the vendor's behavior as disrespectful and unprofessional, leading to frustration and a strained relationship. On the other hand, the vendor sees flexibility as a regular part of managing multiple clients and projects simultaneously. The difference in scheduling expectations leads to misunderstandings and decreased collaboration effectiveness.

These examples highlight the need for open communication about scheduling preferences in software engineering and IT architecture. Architects can foster a more harmonious and productive working environment by recognizing and respecting differing cultural attitudes towards scheduling and finding ways to bridge the gap, such as setting clear expectations and being flexible where possible. This emphasis on open communication encourages us to share our scheduling preferences and understand those of our colleagues, creating a supportive and collaborative work environment.

2.9: Rules

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

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

2.10: 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?*

3: Cooperation-Based Organizations: Six Simple Rules

image by nanostockk from istock

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

KEY POINTS:

- Grounded Architecture emphasizes collaboration and aligns with the Six Simple Rules framework, which promotes autonomy and cooperation to manage organizational complexity effectively.
- The Six Simple Rules reject traditional hard and soft management styles, instead advocating for structures that empower people while encouraging mutual accountability and cooperation.
- Key rules include understanding actual work behaviors, reinforcing integrators, and increasing the total power in the organization to enable decision-making without creating bureaucracy.
- Cooperation is strengthened through reciprocity, feedback loops ("shadow of the future"), and rewarding collaborative behavior, creating a system where shared success drives performance.
- Architecture practices play a vital role by using tools like Lightweight Architectural Analytics and Collaborative Networks to support these rules and foster an adaptive, high-impact environment.

The Grounded Architecture operating model is **fundamentally collaborative**. It focuses on establishing **organizational structures based on cooperation**, allowing architects to work effectively with others and make a meaningful impact. To foster such a collaborative approach, I have found the Six Simple Rules method, developed by Yves Morieux and Peter Tollman, to be particularly useful. This framework encourages organizations to adapt to complexity by promoting autonomy and cooperation among their members.

The book *Six Simple Rules: How to Manage Complexity Without Getting Complicated* by Yves Morieux and Peter Tollman has greatly influenced my vision of an architectural practice. The authors introduce the concept of **Smart Simplicity**, outlining six rules or strategies that enable organizations to encourage new behaviors and enhance performance. The Six Simple Rules approach emphasizes that in today's business environment,

it is essential to create **organizational structures based on cooperation**.

Morieux and Tollman developed the Six Simple Rules as a practical solution for today's complex business landscape. The rules in the book rest on the idea that effectively managing complexity requires a **combination of autonomy and cooperation**. They advocate for organizational structures that align with this principle, empowering individuals to act autonomously. This approach is rooted in trusting the skills and capabilities of the organization's people to tackle complex challenges, thereby fostering a cooperative and efficient work environment.

In this chapter, I will explore how Grounded Architecture and the Six Simple Rules complement each other. The ideas from the Six Simple Rules have been a significant source of inspiration for my work. **Conway's Law**[1] illustrates that the relationship between organizational structures and IT architecture is like peanut butter and jelly—stronger and better together. Both the Six Simple Rules approach and architectural practice focus on managing complexity without becoming overwhelmed.

[1] https://martinfowler.com/bliki/ConwaysLaw.html

3.1: Background: Hard and Soft Management

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

3.1.1: Hard Approach

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

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

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

image by cyano66 from istock

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

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

3.1.2: Soft Approach

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

- **Good performance is the by-product of good interpersonal relationships**. Personal traits, psychological needs, and mindsets predetermine people's actions.
- To change behavior at work, you need to **change the mindset (or change the people)**.

image by alessandro biascioli from istock

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

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

needs. This personalization could include customizable dashboards and personalized workspaces.

3.2: Collaboration Approach

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

The Six Simple Rules approach emphasizes that in today's business environment, you must set up **organizational structures based on cooperation**. More specifically, the Six Simple Rules approach involves the interplay of **autonomy** and **cooperation**. The authors emphasize the critical difference between autonomy and self-sufficiency. **Autonomy** is about fully mobilizing our intelligence and energy to **influence outcomes**, including those **we do not entirely control**. **Self-sufficiency** is about **limiting our efforts** only to those outcomes that we **control entirely without depending** on others. Autonomy is essential for coping with complexity; **self-sufficiency** is an obstacle because it **hinders the cooperation** needed to make autonomy effective.

The first three rules create the conditions for **individual autonomy and empowerment** to improve performance.

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

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

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

3.3: Rule 1: Understand What Your People Do

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

image by scyther5 from istock

3.3.1: Guidelines for Understanding Performance

To genuinely understand performance, consider the following principles:

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

3.3.2: Leveraging Architecture Practice

Architecture practice can significantly aid in understanding organizational behaviors through:

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

3.4: 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

3.4.1: Guidelines for Reinforcing Integrators

To strengthen integrators within your organization, consider these strategies:

- **Use emotions** to identify candidates. Feelings provide crucial clues for analysis and can act as symptoms of integration issues.
- **Identify operational units** that can be integrators among peer units due to their specific interests or power.

- **Remove managerial layers** that do not add value and reinforce others as integrators by eliminating specific rules and relying on observation and judgment over metrics when cooperation is involved.

3.4.2: Enhancing Integrators Through Architecture Practice

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

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

3.5: Rule 3: Increase the Total Quantity of Power

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

image by prostock_studio from istock

3.5.1: Guidelines for Increasing Power Quantity

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

- When making design decisions that could shift the balance between central and unit power, functions, and line managers, ensure that some parts of the **organization benefit from new power bases**.

This approach helps meet complexity requirements and avoids future disruptions from pendulum swings.

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

3.5.2: Enhancing Power Quantity Through Architecture Practice

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

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

3.6: Rule 4: Increase Reciprocity

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

image by natnan srisuwan from istock

3.6.1: Guidelines for Enhancing Reciprocity

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

- **Eliminate monopolies** to ensure no single entity has exclusive control.
- **Reduce resources** to encourage more efficient and collaborative use.

- **Create new networks** of interaction to facilitate better communication and cooperation.

3.6.2: Enhancing Reciprocity Through Architecture Practice

Architecture practice can significantly increase reciprocity within an organization:

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

3.7: Rule 5: Extend the Shadow of the Future

The Six Simple Rules approach emphasizes the importance of making visible and clear **what happens tomorrow as a consequence of what they do today**.

image by joe from pixabay

3.7.1: Guidelines for Extending the Shadow of the Future

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

- **Tighten the feedback loop** by increasing the frequency of moments when people experience the consequences of the fit between their contributions.

- **Bring the endpoint forward** by shortening the duration of projects.
- **Tie futures together** so that successful moves are conditioned on contributing to the success of others.
- Ensure people **walk in the shoes they make** for others.

3.7.2: Extending the Shadow of the Future Through Architecture Practice

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

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

3.8: Rule 6: Reward Those Who Cooperate

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

image by stocksnap from pixabay

3.8.1: Guidelines for Rewarding Cooperation

To effectively reward those who cooperate, managers should:

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

- **Avoid blaming failure** and instead focus on blaming the inability to **help or seek help**.
- **Use simple questions** to shift managerial conversations, making transparency and ambitious targets resources rather than constraints. This approach helps managers act as integrators, **leveraging cooperation** and rich information to achieve superior results.

3.8.2: Rewarding Cooperation Through Architecture Practice

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

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

3.9: To Probe Further

- Six Simple Rules: How to Manage Complexity without Getting Complicated[2]

[2]https://www.bcg.com/capabilities/organization-strategy/smart-simplicity

3.10: Questions to Consider

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

4: The Human Side of Decision-Making

image by metamorworks from istock

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

KEY POINTS:

- Decision-making is a human activity subject to human biases and limitations.
- Fundamental biases influencing decision-making include outcome, hindsight, and confirmation biases.
- Human intuition plays a vital role in decision-making.

Cassie Kozyrkov[1], 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. IT architects need to remember that every decision is flavored by the quirks and quibbles of the human mind.

[1] https://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/

4.1: Understanding Human Biases and Limitations

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

4.1.1: Outcome Bias

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

image by mesh cube from istock

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

Suppose we only focus on the outcome, neglecting the rich backstory and the information available at the time of the decision. In that case,

we're akin to a film critic who only watches the ending. This can lead to **misjudging people's abilities**, rewarding or penalizing them based on a volatile mix of luck and skill. Therefore, it's crucial to discern whether someone's success stems from their astute decision-making or sheer chance. When evaluating decisions, it's essential not to be overly dazzled by the outcome.

In IT, outcome bias can come in many different forms, typically leading to the reinforcement of poor practices and processes based on the perceived successful outcome of projects.

Example 1: Ignoring Best Practices Due to Success

- Due to time constraints, a software team decided to release a critical update without performing adequate regression testing. The update has been released, and fortunately, no major issues have been found by the users.
- Because the update did not cause any immediate problems, the team and management might conclude that regression testing is not always necessary, reinforcing a risky behavior based on a lucky outcome rather than sound engineering practices.

Example 2: Overlooking Security Flaws

- A development team delivers a new web application feature without conducting a thorough security review. The feature has gained popularity and has not encountered any immediate security incidents.
- The absence of immediate security breaches leads the team to believe that their approach to security is adequate, even though the feature might have significant vulnerabilities that have not yet been exploited. This can result in a continued lax attitude towards security best practices.

Example 3: Rewarding Speed Over Quality

- A software project is completed and delivered significantly ahead of schedule but with known technical debt and suboptimal code quality. The project is well-received by the client due to its timely delivery.

- Management praises the team for their speed, ignoring the long-term consequences of the technical debt. This could lead to future projects prioritizing speed over quality, increasing the risk of long-term maintenance issues and potential project failures.

Example 4: Skipping Code Reviews

- A development team decides to skip code reviews to save time, and the software is delivered without any major bugs or issues.
- The team concludes that code reviews are not essential, leading to the institutionalization of skipping this critical quality control step. Future projects might suffer from hidden bugs and poor code quality, which could have been caught during code reviews.

Example 5: Misinterpreting Customer Feedback

- A feature is implemented based on limited customer feedback and is launched successfully. The positive reception leads the team to assume their approach to gathering and acting on feedback is effective.
- The team might conclude that extensive user research is unnecessary, believing that limited feedback is sufficient to guide development. This could result in future features missing critical insights from a broader user base, potentially leading to less successful outcomes.

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

4.1.2: Hindsight Bias

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

image by designer491 from istock

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

If you don't **write down this process**, it's like trying to remember a dream—you'll miss many details and be left only with a vague feeling. This makes it tough to judge the quality of your decisions or anyone else's, stalling your growth as a decision-making wizard. Write down your decision-making process. This helps you determine if luck was messing with you or if your skills were on point.

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

Example 1: Technology Stack Decision

- A particular technology stack is chosen for a project, but it later turns out to be ill-suited, causing significant rework.

- After the problems become apparent, developers and stakeholders might claim that it was clear from the beginning that this 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, the perceived advantages of the stack, and the extensive procurement process.

Example 2: Bug Discovery

- A critical bug is discovered in the production environment that causes significant downtime.
- After the bug is found, developers and stakeholders assert that the bug was easy to spot and should have been caught during the testing phase. This perspective disregards the complexity and number of potential issues that testers were dealing with and that the bug was not apparent among the other potential problems at the time.

Example 3: Performance Issues

- A software system experiences performance degradation under high load conditions that were not tested.
- After the performance issues arise, team members and stakeholders might say that it was clear the system would not handle high load and that stress testing should have been prioritized. This perspective ignores that other pressing issues and constraints influenced the original decision-making process.

Example 4: Post-Mortem Analysis

- A software project fails due to unexpected integration issues with third-party services.
- During the post-mortem analysis, team members and management claimed that the integration issues were obvious and should have been anticipated. They overlooked that the integration appeared straightforward at the time of decision-making, and the issues were not foreseeable with the information available.

Example 5: Feature Failure

- A new feature is released but fails to gain user adoption, which is considered a failure.
- Team members and management might claim that they always had doubts about the feature's success and that it was destined to fail. This can lead to the erroneous belief that the failure was evident from the start, even if the decision to proceed with the feature was based on thorough research and positive initial feedback.

Example 6: Security Breach

- A security vulnerability is exploited in a production system, leading to a data breach.
- Post-breach, it is often stated that the vulnerability was obvious and should have been addressed sooner. This belief neglects the context in which security measures were evaluated and the myriad of potential vulnerabilities that needed to be managed simultaneously.

Example 7: Project Timeline Overruns

- A software project exceeds its deadline due to unforeseen technical challenges.
- Once the challenges are known, team members might argue that the delays were predictable and that the original timeline was overly optimistic. This view ignores the uncertainty and the incomplete information available when the original timeline was set.

By acknowledging the role of hindsight bias, teams can foster a more realistic and fair evaluation of past projects and decisions.

4.1.3: Confirmation Bias

Confirmation bias is like having a pair of magical glasses that only let you see what you already believe. When you stumble upon a new fact, your brain gives it a makeover to fit your beliefs, even before your morning coffee.

image by designer491 from istock

Awareness of this sneaky bias is essential because your brain loves playing tricks on you. It makes you think you're being objective while sneakily **reinforcing your pre-existing ideas**. This subconscious nudge can twist your understanding and decision-making without you even noticing.

Businesses are jumping on the data science bandwagon, hiring data scientists to make supposedly unbiased, data-driven decisions. But guess what? These decisions are often not as data-driven as they claim. For a decision to follow the data, it should be the data leading the dance, not your preconceived notions or biases—a simple idea harder to pull off than a flawless magic trick.

Confirmation bias is like your brain's way of playing a one-sided game of telephone with data. Even the most complex math won't save you if you still interpret results through your belief-tinted glasses. Extensive data analysis can be as helpful as a screen door on a submarine if it's warped by confirmation bias. The real challenge is resisting the urge to twist the story after seeing the data. So, watch for your brain's tricks and let the data speak for itself—or risk your analysis being as misleading as a carnival funhouse mirror.

Beating confirmation bias is like prepping for a magic show: you need a plan before the curtain goes up. **Set clear objectives** before you peek

at the data. Consider what the data should mean to you beforehand so you don't get dazzled by surprising plot twists. This way, you can make genuinely data-driven decisions instead of falling into the trap of your brain's sneaky biases.

Confirmation bias in IT can lead to suboptimal decisions and hinder the effectiveness of problem-solving and innovation in IT:

Example 1: Tool Selection

- A development team prefers using a specific development framework because of their past experiences with it.
- When choosing a framework for a new project, team members only seek out positive reviews and success stories about their preferred framework, ignoring or downplaying any negative feedback or alternative frameworks that might be better suited for the project's requirements.

Example 2: Technology Adoption

- A company decides to adopt a new technology stack based on industry trends and some initial positive experiences.
- The team focuses on success stories and favorable benchmarks supporting the decision while disregarding case studies or reports highlighting challenges and failures associated with the new technology. This can lead to underestimating the risks and difficulties of the adoption process.

Example 3: Debugging

- A developer believes that a particular module is the source of a bug.
- The developer focuses exclusively on the suspected module, interpreting any evidence to support this belief, and may overlook or disregard indications that the bug originates from another part of the code. This can lead to extended debugging time and potentially missing the actual source of the issue.

Example 4: Performance Testing

- A team is confident that their application will perform well under high load due to recent optimizations.

- When conducting performance tests, they may primarily focus on scenarios where they expect the application to perform well, ignoring or not thoroughly testing edge cases or scenarios that might reveal performance bottlenecks. As a result, they might miss critical issues that only appear under certain conditions.

Example 5: Estimating Project Timelines

- A project manager strongly believes the team can meet an aggressive deadline.
- The project manager seeks out and emphasizes information and past examples where similar deadlines were met while ignoring or discounting instances where similar projects encountered delays. This can lead to unrealistic project timelines and potential burnout.

Example 6: Code Reviews

- A senior developer has a high opinion of a specific junior developer's skills.
- During code reviews, the senior developer may be more inclined to approve the junior developer's code with minimal scrutiny, interpreting any ambiguities or minor issues as acceptable or easily fixable, while being more critical of other developers' code.

Example 7: User Feedback

- The team has a preconceived notion that users will love a new feature they developed.
- When gathering user feedback, they may give more weight to positive comments and downplay or dismiss negative feedback. They might also ask leading questions likely to elicit positive responses, thus reinforcing their belief that the feature is well-received.

To mitigate the impact of confirmation bias, teams should actively seek out and consider disconfirming evidence, adopt a critical thinking approach, and encourage diverse perspectives. By being aware of this bias, teams can make more balanced and informed decisions, ultimately leading to better software outcomes.

4.1.4: Other Human Limitations Influencing Decision-Making

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

image by aaronamat from istock

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

And let's be honest—your decision-making skills aren't precisely Olympic-level when you're **sleep-deprived**, **hungry**, **stressed**, or feeling

the heat. These biological and emotional states can affect your ability to make top-notch decisions. Believing that sheer willpower or a PhD in decision-making will always lead to the best outcomes is like thinking you can win a marathon in flip-flops.

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

4.2: Understanding Power and Limitations of Human Intuition in Decision-Making

Human intuition plays a vital role in decision-making. Robert Glass provided one of the best definitions of intuitions, describing it as a function of our mind that allows it to access **a rich fund of historically gleaned information** we are not necessarily aware we possess by **a method we do not understand** (Glass, 2006; page 125)[4]. But oOur unawareness of such knowledge does not mean we cannot use it.

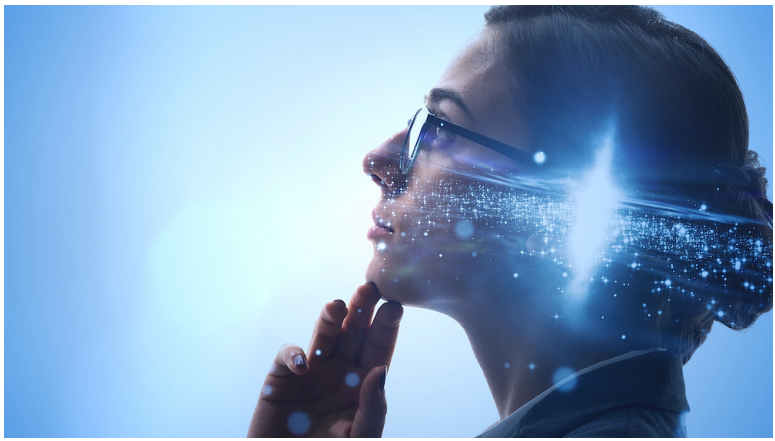

image by metamorworks from istock

One of the main advantages of intuition in decision-making is that accessing it is a rapid process, making intuitive decisions straightforward. Intuition is particularly useful for low-value decisions with low stakes, and a quick resolution is preferable. As we'll explore in future discussions on prioritization and decisiveness, seeking perfection in every decision is impractical due to limited time and energy. Therefore, it's essential to choose where to focus your efforts.

Intuition is especially appropriate under certain conditions:

- **Time Pressure**: When time is limited and a detailed analysis isn't

[4]https://www.amazon.com/Software-Creativity-2-0-Robert-Glass/dp/0977213315

feasible, intuition can guide you when otherwise you would be stuck.

- **Expertise**: If you have experience in a particular area, relying on intuition makes sense, as you've likely faced similar decisions before. In contrast, in unfamiliar contexts, intuition may not be reliable.
- **Unstructured Decisions**: Intuition can be valuable for decisions that lack a clear framework, like judging the quality of art. Expertise in the relevant field enhances the effectiveness of intuitive judgments.

Conversely, you should avoid relying on intuition too much when more effort is warranted, including those with ample time, high importance, lack of expertise, and the possibility to use a structured decision-making process.

Intuition in IT can be a **powerful tool** when informed by experience and used in conjunction with data and thorough analysis. It can guide efficient problem-solving and quick decision-making in familiar contexts. However, overreliance on intuition without considering empirical evidence and diverse viewpoints can lead to **significant mistakes**, especially in unfamiliar or complex situations.

4.2.1: Good Example: Intuitive Debugging

Scenario: A seasoned software engineer is working on a complex system that suddenly starts behaving unexpectedly. The logs provide little information, and initial debugging efforts do not yield apparent clues.

Good Use of Intuition

1. **Pattern Recognition**: Drawing on years of experience, the engineer intuitively suspects that the issue might be related to a recent configuration change rather than a code issue.
2. **Focused Investigation**: Based on this intuition, the engineer quickly narrows down the potential causes, focusing on recent changes in the configuration files.
3. **Quick Resolution**: This intuition-driven approach reveals a misconfiguration in minutes, saving the team hours of potentially fruitless debugging.

Outcome: The engineer's intuition, honed by experience, helps quickly identify and resolve the issue, demonstrating how intuition can efficiently guide problem-solving in complex scenarios and under time pressure.

4.2.2: Bad Example: Intuitive Decision on Technology Stack

Scenario: A new project is starting, and the team needs to decide on the technology stack. The team lead has a strong intuitive preference for a specific new programming language and framework they have used.

Bad Use of Intuition

1. **Ignoring Data**: The team lead dismisses team members' concerns and data about the scalability and community support of the chosen technology. There was ample time to do proper analysis.
2. **Overconfidence**: Relying solely on personal intuition and past experience, the team lead pushes forward with the technology despite its known limitations for the project's specific needs.
3. **Future Problems**: As the project progresses, the team encounters significant issues related to performance and maintainability. These issues could have been mitigated or avoided by choosing a more appropriate technology stack based on objective criteria and thorough evaluation.

Outcome: The team lead's overreliance on intuition leads to a poor technology choice, resulting in increased technical debt, reduced productivity, and ultimately a less successful project. This example highlights how intuition can lead to suboptimal decisions when used without adequate consideration of data and other perspectives.

Balancing intuition with data-driven decision-making and collaborative input often leads to the best outcomes in software engineering.

4.3: Understanding Human Indecisiveness

We frequently fall into the indecisiveness trap. Why? Well, people can be indecisive for all sorts of amusing reasons.

4.3.1: Delaying Decisions

Many folks don't realize that **dodging a decision is still a decision**. It's like standing in front of an ice cream shop, unable to choose a flavor until the shop closes, and boom—you've "decided" to go home without ice cream. Delaying, postponing, or deprioritizing the decision-making process is an implicit choice.

image by dragon claws from istock

Dodging decisions often lead to significant consequences, even if they aren't immediately obvious. Here are a few examples:

Microservices vs. Monolithic Architecture

- **Scenario**: A team is debating whether to move to a microservices architecture or continue with their existing monolithic system.

Rather than making a decision, they keep delaying it, hoping that some perfect clarity will emerge.
- **Implicit Decision**: The team effectively "decides" to stay with their monolithic system by not choosing. Over time, this makes it harder to scale, introduces more technical debt, and slows deployment and innovation. They've made a choice—whether they realize it or not.

Code Refactoring vs. Feature Development

- **Scenario**: Engineers know that a particular part of the codebase needs refactoring, but with each sprint, the refactoring task gets postponed in favor of new features.
- **Implicit Decision**: By deprioritizing refactoring, the team implicitly decides to live with a growing pile of technical debt, leading to more bugs, slower performance, and increasingly difficult maintenance.

Cloud Migration

- **Scenario**: A company wants to move its infrastructure to the cloud but keeps postponing the decision, citing the need for further research or budgetary constraints.
- **Implicit Decision**: By delaying the migration, they implicitly decide to continue using outdated on-premise systems, which may be less efficient and more expensive in the long run. They also miss out on the agility and scalability benefits of cloud infrastructure.

Testing Strategy

- **Scenario**: A team debates about the scope of automated vs. manual testing. However, because they can't decide, they continue relying on an inconsistent mix of both without a comprehensive strategy.
- **Implicit Decision**: The lack of a solid testing strategy leads to reduced code quality, longer release cycles, and more defects slipping into production. The choice to not formalize testing becomes a costly implicit decision.

Single Sign-On (SSO) Integration

- **Scenario**: A company debates whether to implement SSO for its internal systems to streamline user authentication. However, security and engineering teams defer the decision to focus on more "urgent" projects.
- **Implicit Decision**: By not implementing SSO, the company has multiple login systems, causing frustration for users and increased security risks due to inconsistent authentication methods.

In all of these cases, failing to make a decision is itself a decision—and often a costly one. The key takeaway here is that avoiding decisions in technical scenarios can lead to significant consequences, even if they aren't immediately obvious.

4.3.2: Overwhelmed by Numerous Decisions

Then there's the classic "**too many choices**" dilemma. When faced with an array of decisions, especially those of lower priority, our brains feel like an overstuffed suitcase. Our cognitive capacity is limited—we can't focus intensely on everything simultaneously. Suppose we spend too much mental energy deciding what color to paint the break room. In that case, we might leave no brainpower for the big decisions, like how to keep the servers from catching fire.

image by cyano66 from istock

The "too many choices" dilemma is a common problem where teams or decision-makers become overwhelmed by many low-priority decisions, often detracting from critical, high-impact choices. Here are a few examples:

Library and Dependency Updates

- **Scenario**: Developers ask the IT architect to review and approve every minor version upgrade for third-party libraries, even those that involve minimal risk or functionality changes.
- **Problem**: By focusing the architect's attention on these low-impact updates, critical decisions (like system scalability, security measures, or architectural design) may get delayed. It's like asking the architect to approve every paint stroke while the house's foundation needs reinforcement.

Code Formatting Standards

- **Scenario**: The engineering team spends excessive time debating the best code formatting or style guidelines, like whether to use 2 or 4 spaces for indentation.

- **Problem**: While consistency in code formatting is important, overallocating time to decisions like these diverts attention from more significant decisions, such as improving the application's architecture or fixing major bugs. It's like worrying about the snacks in the break room while production servers are down.

Platform or Tool Selection

- **Scenario**: The team chooses between different code editor plugins, build tools and minor productivity tools. Each option is reviewed in detail, with countless meetings to compare feature lists.
- **Problem**: Overanalyzing small tooling decisions causes unnecessary delays and burns mental energy that is better spent on high-impact decisions, such as selecting a cloud provider or deciding the proper microservices orchestration approach.

Server Naming Conventions

- **Scenario**: The IT department holds extensive discussions on naming the new servers or virtual machines in the cloud environment—should they use city names, numbers, or specific animal species?
- **Problem**: This seemingly minor debate consumes valuable time and energy, while more critical infrastructure decisions, like disaster recovery planning or server load balancing, get delayed or ignored.

Approving Minor Configuration Changes

- **Scenario**: IT architects are asked to approve every minor configuration tweak, such as adjusting the timeout for a service or tweaking memory limits for non-critical applications.
- **Problem**: The architects drown in trivial decisions, leaving them little bandwidth to focus on higher-priority issues, like designing robust, scalable system architectures or improving application security.

In these examples, the team's limited cognitive capacity is consumed by trivial choices, leaving insufficient mental energy for critical decisions that genuinely impact system performance, security, or scalability. This issue highlights the importance of delegating or automating low-priority decisions and reserving strategic brainpower for what truly matters.

4.3.3: Emotions and Grief

Indecisiveness also loves to rear its head when emotions are involved, especially when all options are as appealing as a soggy sandwich. When faced with **undesirable choices**, the practical move is to pick the **least bad option**. After thoroughly evaluating your choices and identifying the least awful one, it's time to bite the bullet and make the call.

People often get tangled in a web of **grief or frustration** when stuck with lousy options, hoping for a miracle solution that'll never come. It's like searching for unicorns in a petting zoo. Once it's clear that no better options will appear, it takes courage to move forward with the least terrible choice.

image by violetastoimenova from istock

Teams often face situations where every option feels undesirable, and indecisiveness can creep in due to emotional frustration or the hope for a perfect solution. Here are some examples:

Legacy System Migration

- **Scenario**: A company runs critical applications on a legacy system

that is slow, costly to maintain, and increasingly prone to failure. The options are to either:

1. Perform a risky, costly, and time-consuming migration to a modern system.
2. Continue investing in patching the legacy system, with the risk of a major outage.

- **Emotional Factor**: No one wants to be responsible for the potential failures of a complex migration, but continuing with the legacy system feels like kicking the can down the road.
- **Least Bad Option**: After evaluating the long-term costs and risks, the team decides to proceed with the migration, accepting the immediate pain of potential downtime and cost over the unsustainable alternative of maintaining the outdated system. It's not a great choice, but it's the least bad one, so they bite the bullet.

Technical Debt Management

- **Scenario**: A project has accumulated significant technical debt. The options are:

1. Allocate several sprints to pay the debt, delaying new feature development.
2. Continue adding features on top of a shaky codebase, which will inevitably slow development and increase the risk of defects.

- **Emotional Factor**: Teams often feel frustrated by technical debt, wishing it would somehow resolve itself. However, neither option is appealing—both involve trade-offs between immediate feature delivery and long-term stability.
- **Least Bad Option**: Recognizing that continued development on a brittle foundation will lead to far greater problems, the team accepts the frustration of delaying features and tackles the technical debt, understanding it's the least bad way to ensure future scalability and maintainability.

Vendor Lock-In

- **Scenario**: A company is heavily invested in a particular cloud provider but realizes the cost is much higher than anticipated. The choices are:

    1. Stay with the current provider and absorb the higher costs.
    2. Undertake a complex and costly migration to a new provider, risking downtime and re-engineering efforts.

- **Emotional Factor**: Switching vendors feels overwhelming and risky, while staying locked into an expensive provider feels frustrating. Hoping for a "magic" solution (e.g., the provider reducing costs) isn't realistic.
- **Least Bad Option**: After weighing the costs and risks, the company decides to stay with the current vendor, accepting the cost over the uncertainty of migration. It's not ideal, but it's the least bad choice given the alternatives.

Monolithic vs. Microservices Split

- **Scenario**: A monolithic application has grown too large, and the team knows it needs to break it into microservices. The options are:

    1. Undertake a complex, high-risk rewrite of the monolithic application into microservices.
    2. Continue maintaining the monolith, accepting slower release cycles and scaling limitations.

- **Emotional Factor**: Rewriting a system always feels daunting, and no one wants to own the risk of disrupting the system during the transition. However, maintaining the monolith comes with its own set of frustrations, especially as it grows.
- **Least Bad Option**: Recognizing that continuing with the monolith will lead to stagnation, the team opts for a phased approach to break the system into microservices. It's a difficult and risky path, but the alternative of doing nothing is worse.

Security Fixes vs. Feature Development

- **Scenario**: A critical vulnerability is discovered in an application, but fixing it will require a full sprint of engineering time, delaying a high-priority feature release.

    1. Delay the feature and fix the vulnerability immediately, which will frustrate stakeholders and users awaiting the feature.
    2. Continue with the feature release and address the security issue later, increasing the risk of a breach.

- **Emotional Factor**: The team is pressured by stakeholders eager for new features, but the security risk is causing anxiety. Neither option is appealing, as both involve security and feature delivery trade-offs.
- **Least Bad Option**: The team addresses the security issue first, accepting the frustration of delaying the feature because the consequences of a breach are far worse. It's not a great situation but the least bad decision.

In each case, the choices are less than ideal, and emotions like frustration, anxiety, or fear of failure can cloud judgment. However, teams can move forward with clarity and purpose by acknowledging that no perfect option exists and focusing on the least detrimental path.

4.4: Understanding Group Dynamics in Decision-Making

Effective decision-making often involves recognizing that **you might not be the sole decision-maker**. In organizations, it's crucial to identify the actual decision-makers and understand how decision responsibility is distributed among them. Mastering this skill is essential for navigating organizational decision-making processes. It's important to question who really has the final say in decisions. In many cases, decision-making is more complex than it appears.

image by prostock_studio from istock

Group decision-making offers significant advantages. While you might believe you have the best solutions, incorporating diverse perspectives can help cover your blind spots. Multiple decision-makers can counterbalance an individual's extreme tendencies and compensate for human limitations like fatigue.

4.4.1: Characteristics of Group-Decision Making

While group decision-making might sometimes constrain individual creativity, it also **provides safeguards against poor decisions** and **aligns**

individual motives with the organization's goals. Having several independent decision-makers can align individual incentives with the organization's needs, addressing this problem.

However, group decision-making isn't perfect. It **increases complexity** as it **requires higher decision-making skills** from each member. True **collaboration in decision-making** is more challenging than individual decision-making. It also tends to **slow down the decision process**.

Moreover, the benefits of group decision-making, like balancing individual biases, **rely on the independence of the decision-makers**. If everyone is in the same room, independence can be compromised by factors like **charisma or status**, potentially allowing the loudest voice to dominate rather than the wisest.

Group settings can also **devolve into social exercises**, where **personal ego overshadows open-mindedness** to new information. Awareness of these pitfalls allows you to create rules that foster independent perspectives.

The **role of the note-taker** in group settings is also influential, as is the phenomenon of **responsibility diffusion**, where **unclear responsibilities** lead to reduced individual contribution.

In summary, **the more people involved in a decision, the higher the skill level required** to maximize the benefits and minimize the downsides of group decision-making. It's vital to structure the process to maintain independence, possibly by limiting decision-makers and increasing advisors. This approach distinguishes between making a decision and advocating for the execution of an already-made decision.

4.4.2: Examples

Group decision-making dynamics in IT can take various forms, including consensus, hierarchical, voting, and conflict resolution approaches. Group decision-making dynamics in IT can vary widely depending on the context, team structure, and decision at hand. Here are some examples illustrating different aspects of group decision-making dynamics in IT:

4.4.2.1: Example 1: Consensus Decision-Making for Technology Adoption

Scenario: An IT team must decide which cloud platform to use for a new project. The options are AWS, Azure, and Google Cloud.

Dynamics:

1. **Information Sharing**: Team members share their experiences and knowledge about each platform. This includes presenting pros and cons, costs, and performance benchmarks.
2. **Brainstorming**: An open discussion is held where everyone is encouraged to voice their opinions and suggest potential solutions.
3. **Evaluation**: Each option is evaluated based on predefined criteria such as scalability, cost, ease of integration, and existing team expertise.
4. **Consensus Building**: The team works towards a consensus by discussing the trade-offs and attempting to agree on the platform that best meets the project's needs.
5. **Decision**: After a thorough discussion, the team decides to use AWS due to its robust ecosystem and familiarity with it.

Influence: Consensus decision-making ensures that all team members feel heard and can contribute to the decision, leading to higher buy-in and commitment to the chosen platform.

4.4.2.2: Example 2: Hierarchical Decision-Making for Security Policy

Scenario: A decision must be made about implementing a new security policy to comply with regulatory requirements.

Dynamics:

1. **Top-Down Directive**: Senior management decides on the necessity of the new security policy based on compliance needs and risk assessments.
2. **Expert Input**: Security experts within the organization are consulted to provide detailed recommendations on implementing measures.

3. **Implementation Plan**: The IT manager creates an implementation plan based on the expert recommendations and communicates it to the team.
4. **Team Execution**: The IT team is tasked with executing the plan, following the directives provided by management.

Influence: Hierarchical decision-making can be efficient, especially when quick, decisive action is required and the decision involves specialized knowledge. However, it may result in less buy-in from the team if they are not involved in the decision-making process.

4.4.2.3: Example 3: Voting for Feature Prioritization

Scenario: A software development team needs to prioritize features for the next release of their product.

Dynamics:

1. **Feature List**: A list of potential features is compiled based on customer feedback, market research, and internal brainstorming sessions.
2. **Discussion**: The team discusses the importance and impact of each feature, considering factors such as user value, development effort, and strategic alignment.
3. **Voting**: Each team member votes on their top features, often using a point system where they can allocate a certain number of points across the features.
4. **Ranking**: Features are ranked based on the total points received, and the top-ranked features are selected for the next release.

Influence: Voting democratizes the decision-making process and ensures that the prioritization reflects the team's collective opinion. This approach can enhance team morale and provide diverse perspectives are considered.

4.4.2.4: Example 4: Conflict Resolution in Architecture Decisions

Scenario: The development team is divided over whether to build a new application using a microservices or monolithic architecture.

Dynamics

1. **Initial Positions**: Team members present their initial positions, with some advocating for microservices due to their scalability and flexibility and others for a monolithic architecture due to its simplicity and ease of deployment.
2. **Evidence Gathering**: Both sides present evidence, including case studies, technical articles, and expert opinions, to support their arguments.
3. **Facilitated Discussion**: A neutral facilitator (such as an architect) leads a structured discussion to explore the pros and cons of each approach.
4. **Compromise and Integration**: The team seeks a compromise or an integrated solution, such as starting with a monolithic architecture and planning to evolve to microservices as the application grows.
5. **Final Decision**: After thoroughly discussing and considering all viewpoints, the team decides to balance immediate needs with future scalability.

Influence: Structured conflict resolution ensures that all voices are heard and helps the team make a well-considered decision. Combining the strengths of different viewpoints can enhance mutual understanding and lead to better decisions.

Each method has its advantages and can be suitable for different decisions. Understanding these group dynamics can help teams navigate complex choices more effectively, leading to better outcomes and stronger team cohesion.

4.5: Questions to Consider

- *How do your personal biases, such as outcome, hindsight, and confirmation bias, influence your decision-making process? Can you identify a recent decision where these biases might have played a role?*
- *Reflect on a situation where the outcome was bad, but the quality of your decision-making process was solid. How did you respond to this outcome, and what lessons did you learn?*
- *In what ways do you think hindsight bias has affected your ability to evaluate past decisions accurately? Can you think of a decision that seemed obvious in retrospect but was unclear at the time?*
- *Consider a decision you made recently. Did you document the decision-making process? If not, how could documenting this process help you in evaluating your decisions more effectively in the future?*
- *How does confirmation bias impact your interpretation of new information? Can you recall an instance where you ignored or misinterpreted data to fit your pre-existing beliefs?*
- *Think about a decision where changing your perspective or how information was presented (e.g., through different wording) might have led you to a different conclusion. How does this realization affect your approach to decision-making?*
- *Can you identify any habits or emotional factors contributing to your indecisiveness? What strategies can you employ to overcome these challenges?*
- *Reflect on a time when your physical or emotional state might have influenced a decision. What does this tell you about the importance of being aware of your condition when making decisions?*
- *Consider a decision where intuition played a significant role. Was the decision effective, and would you rely on intuition under similar circumstances in the future?*
- *How do you balance the benefits of group decision-making with its challenges, such as social dynamics and the diffusion of responsibility?*

5: Effortless Architecture

image by chinnapong from istock

IN THIS SECTION, YOU WILL: Get a summary of lessons learned from Greg McKeown's book Effortless on how to functionally structure your work to make the essential activities the easiest ones to achieve.

KEY POINTS:

- Greg McKeown's "Effortless: Make It Easier to Do What Matters Most" advocates for a paradigm shift from hard work to smart, effective work by simplifying tasks and processes.
- Key principles include prioritizing important tasks, leveraging automation, and embracing a mindset that values ease and enjoyment in work.
- Greg McKeown's book offers invaluable insights that are particularly relevant for IT architects and software engineers. McKeown's emphasis on simplifying tasks and processes is crucial in the tech industry, where complexity often dominates.

Greg McKeown's "**Effortless: Make It Easier to Do What Matters Most**[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 importance of reducing **accidental complexity**—those unnecessary complications we can eliminate to streamline processes and simplify tasks we complicate ourselves. While Brooks highlights the unavoidable challenges intrinsic to software development, McKeown offers a crucial reminder that streamlining and optimizing workflows can significantly reduce extraneous difficulties,

[1] https://gregmckeown.com/books/effortless/
[2] https://en.wikipedia.org/wiki/No_Silver_Bullet

thus enhancing overall efficiency and effectiveness. Or, as McKeown put it *"life doesn't have to be as hard and complicated as we make it."*

5.1: IT Doesn't Have To Be As Hard and Complicated As We Make It

One of the main tasks of architects is to remind everyone that our technical designs, products, and organizational structures don't have to be as complex and complicated as we make them. A fantastic example of this comes from Pragmatic Dave Thomas (I heard this anecdote on the SE-Radio podcast with Neal Ford[3], ~32 minutes in). A company had problems with its mail post not getting to the proper departments. It wanted a complex optical-character recognition (OCR) system for routing mail posts. However, Dave Thomas suggested using simple and cheap colored envelopes instead. The company did not need to invest millions in building a complex software system with machine-learning capabilities; it solved the problem in a few weeks and saved a lot of money.

Similarly, Greg McKeown's book summarizes many well-known practices into a pragmatic framework with a mindset of effortlessness. As such, I have found that it offers a fresh look at the daily practice of IT architects and software engineers:

- **Simplifying tasks** and processes is crucial in the tech industry, where complexity often dominates. Effortless principles align closely with critical system design and coding practices, such as modular design, clean code, and lean architecture. IT professionals can significantly enhance efficiency by breaking complex systems into manageable modules, writing maintainable code, and focusing on essential features without over-engineering.
- **Prioritization**, another core aspect of McKeown's book, is vital in the fast-paced IT industry. Effectively prioritizing tasks can dramatically impact project outcomes, helping professionals focus on what truly matters. This prioritization leads to more effective project management, resource optimization, and strategic planning, aligning development efforts with business goals.
- **Efficiency and productivity** are also central themes in "Effortless." For software engineers, this translates to practices like automated testing and deployment through CI/CD pipelines, optimization

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

techniques to enhance code performance, and using development tools that streamline workflows.

- McKeown's advocacy for shifting from hard work to smart work is particularly pertinent in the tech world. This mindset shift promotes **continuous learning**, a healthy **work-life balance**, and **resilience strategies** to handle challenges positively.
- **Collaboration and communication**, highlighted in "Effortless," are essential for IT professionals. They enhance collaboration between development, operations, and business teams and ensure stakeholder engagement, leading to more aligned and informed teams.
- In high-pressure environments like the tech industry, **managing stress** is crucial. McKeown's strategies for reducing unnecessary effort and anxiety can help IT professionals focus on high-value tasks, improve mental health, and boost team morale, creating a more enjoyable work experience.

In the following sections, I will review critical advice from McKeown's work, grouped into Effortless State, Effortless Action, and Effortless Results (Figure 1).

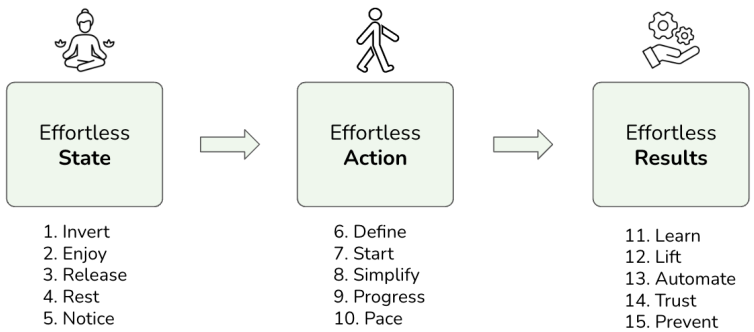

***Figure 1**: The McKeown's Effortless framework: Effortless State, Effortless Action, and Effortless Results.*

5.2: Effortless State

Many have encountered the Effortless State, a **peak experience** when their physical, emotional, and mental well-being align perfectly. You feel physically rested, emotionally unburdened, and mentally energized in this state. You become entirely aware, alert, present, attentive, and focused on what's important. This state allows you to **concentrate on what matters** more quickly and efficiently (Figure 2).

When you achieve the Effortless State, it's a sign that **your brain is operating at its full capacity**. In this optimal condition, tasks that usually feel difficult become significantly easier. You can navigate challenges with a sense of flow and clarity, making decisions and performing actions with a heightened **sense of purpose and precision**. This state enhances your productivity and opens up new avenues for personal growth and development.

However, reaching and maintaining this state can be impeded by **mental clutter**. Clutter can take many forms, including outdated assumptions, negative emotions, and toxic thought patterns. These mental obstacles drain your cognitive resources and make everything feel more complicated than it should be. By clearing this clutter and fostering a supportive mental environment, you can unlock your brain's full potential and access the Effortless State consistently.

***Figure 2**: Effortless State is a part of the McKeown's Effortless framework (State, Action, Results).*

5.2.1: 1. INVERT: What If This Could Be Easy?

Feeling overwhelmed is often not due to a situation's inherent complexity but because we are **overcomplicating it in our minds**. Instead of asking, *"Why is this so hard?"* McKeown proposes to invert the question by asking, *"What if this could be easy?"* Challenge the assumption that the "right" way is necessarily harder. When faced with overwhelming tasks, ask yourself, *"How am I making this harder than it needs to be?"*

By asking, *"What if this could be easy?"* you can reset your thinking. Or, as Kent Beck famously stated, for each desired change, **make the change easy**, **then make the easy change**.

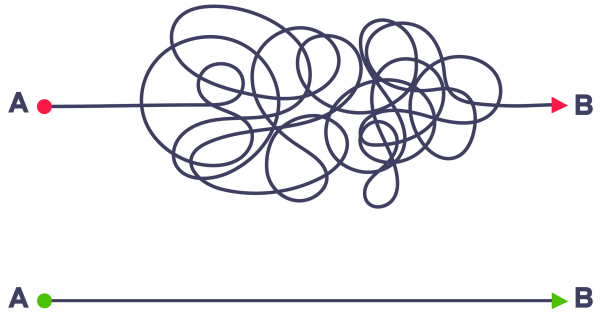

image by mironov konstantin from istock

IT and software engineering have integrated the principles of effortless inversion, transforming many complex tasks into efficient processes. But we must always remind ourselves not to overthink our work.

Here are a few examples that IT architects can use as an inspiration to help their organizations make things easier:

- **Simplifying Code Maintenance With Clean Design**: Rushing to build new features by creating shortcuts and tech debt is a recipe

for overcomplicating your future code maintenance. Creating a modular source code design with clear, simple, and well-tested functions makes code maintenance easier. Clean and modular code leverages existing libraries, prioritizing readability and maintainability over micro-optimizations. These widely known practices simplify maintenance, reducing the time and effort needed for debugging and updates.

- **Streamlining Deployment Processes**: Ad-hoc manual processes are a frequent source of unnecessary complexity in software engineering. Teams implement Continuous Integration/Continuous Deployment (CI/CD) pipelines using tools like Jenkins, GitHub Actions, or GitLab CI. These pipelines automate testing, building, and deployment processes, significantly reducing the potential for human error. This not only expedites the deployment process but also makes it more reliable, ensuring a smoother and more efficient workflow.
- **Simplifying User Interface (UI) Design With Standard Frameworks**: Using UI frameworks like Material Design to create consistent and straightforward interfaces significantly enhances the user experience while simplifying development. A standard UI reduces the cognitive load on developers and end-users and makes the application more straightforward to maintain, also leading to higher user satisfaction.

5.2.2: 2. ENJOY: What If This Could Be Fun?

Combining essential tasks with pleasurable activities can enhance your productivity while maintaining a sense of well-being. Accept that it is possible and beneficial to **integrate work and play**. Pair the most essential activities with the most enjoyable ones. Embrace the idea that work and play can co-exist harmoniously. Transform tedious tasks into meaningful rituals, infusing them with purpose and enjoyment. Allow laughter and fun to lighten more of your moments, turning routine activities into opportunities for joy and creativity.

This approach helps you stay engaged and motivated, making even the most mundane tasks feel more fulfilling and less burdensome. Letting joy and laughter permeate your daily routine creates a positive, dynamic environment where productivity and happiness thrive together.

image by azmanl from istock

Pairing essential activities with enjoyable ones creates a harmonious and engaging work environment for IT and software engineering professionals. This approach enhances productivity, promotes well-being, and increases job satisfaction by turning routine tasks into opportunities for joy and creativity.

Here are a few examples that IT architects as an inspiration to help their organizations integrate work and play:

- **Architecture Reviews as Collaborative Learning Sessions**: Developers often see architecture reviews as bureaucratic, tedious, and time-consuming, but you can transform them into collaborative learning sessions. Adding a gamification element, such as rewarding the most insightful feedback, can make such reviews social and enjoyable. This engagement leads to better designs and stronger team cohesion.
- **Writing Technical Documentation**: Technical documentation, often perceived as monotonous, can be made more enjoyable by infusing creativity. Using storytelling techniques or incorporating visual elements like diagrams, infographics, and comic strips can

make the task engaging. Encouraging personal anecdotes or humor where appropriate results in more comprehensive and user-friendly documentation, making the process enjoyable for both writers and readers.

- **Keeping Up with New Technologies**: Continuous learning and keeping up with new technologies can feel overwhelming and exhausting. Aligning learning activities with personal interests and hobbies helps maintain motivation and enthusiasm. For instance, exploring game development or gamification techniques for team members who enjoy gaming, or delving into UI/UX design trends for those who love graphic design, makes continuous learning more enjoyable and fulfilling.

5.2.3: 3. RELEASE: The Power Of Letting Go

As the saying goes, ***"the best thing one can do when it is raining is to let it rain."*** By acknowledging and embracing our circumstances, we can focus on what we have rather than what we lack, fostering gratitude and emotional resilience.

Regrets that continue to haunt us, grudges we can't seem to let go of, and expectations that once were realistic but now hinder us all contribute to our emotional burdens. When we fall victim to misfortune, we easily obsess, lament, or complain about all we have lost. **Complaining** is one of the easiest things to do.

image by mixmike from istock

Similarly, IT and software engineering professionals can release unnecessary **emotional burdens**, fostering a positive and resilient mindset that enhances personal well-being and professional effectiveness.

Here are a few examples that IT architects as an inspiration to help their organizations let go:

- **Overcoming Regret Over Past Decisions**: Engineers and IT architects may regret choosing a particular technology stack or making architectural decisions that later proved suboptimal. Accept that decisions were likely made with the best knowledge available at the time and focus on learning from past experiences rather than dwelling on them. When regret surfaces, reflect on a positive outcome or lesson learned from that decision and express gratitude for the growth it provided. Shifting focus from regret to learning fosters a growth mindset and empowers engineers to make informed decisions without being weighed down by past mistakes.
- **Letting Go of Grudges Against Team Members**: Holding grudges against colleagues for past misunderstandings or conflicts can create a toxic work environment. Address and resolve conflicts through open communication and empathy, recognizing that holding grudges serves no constructive purpose. Each time a grudge

resurfaces, consciously let it go by acknowledging the colleague's positive trait or contribution. Releasing grudges promotes a harmonious and collaborative team environment, improving morale and productivity.

- **Embracing Change and Letting Go of Resistance**: Resistance to adopting new technologies or methodologies can hinder progress and innovation. Be open to change and willing to experiment with the latest tools and approaches, viewing change as an opportunity for growth rather than a threat. When encountering resistance to change, identify one positive aspect or potential benefit of the new approach and express gratitude for the opportunity to innovate. Embracing change fosters innovation and keeps the team adaptable and forward-thinking, driving continuous improvement and success.

5.2.4: 4. REST: Take a Break

By incorporating periods of **rest and reflection**, you create a balanced routine that enhances productivity and overall quality of life. This structured approach allows you to maintain high levels of energy and concentration, preventing burnout and ensuring that you can consistently perform at your best. Embrace the art of doing nothing. Avoid overextending yourself by not doing more today than you can fully recover from by tomorrow. This approach promotes sustainable productivity and well-being.

image by chinnapong from istock

By embracing the art of doing nothing and incorporating structured work sessions, rest periods, and balanced routines, IT and software engineering professionals can enhance their productivity and well-being. This approach promotes sustainable productivity, prevents burnout, and ensures that engineers can consistently perform at their best.

Here are a few examples that IT architects as an inspiration to help their organizations take a break:

- **Incorporating Rest and Reflection Periods**: Engineers frequently move from one task to another without taking time to reflect and rest, leading to mental fatigue. Scheduling short breaks after each focused session for rest and reflection can mitigate this. Activities such as walking, meditating, or simply quietly sitting can be beneficial. For instance, after each 90-minute session, a 15-minute walk outside or 10 minutes of meditation can help clear the mind. These regular breaks help maintain mental clarity and focus, enhancing overall productivity and well-being.
- **Balancing Workload and Recovery**: Taking on more work than one can recover overnight leads to cumulative fatigue and stress for engineers. Effectively managing the workload ensures daily tasks are balanced and realistic, avoiding overextension. Tools like

task lists and time-blocking can help manage workload effectively. Prioritizing tasks and allocating time based on their complexity and importance while avoiding scheduling back-to-back high-intensity tasks ensures recovery and consistent performance, preventing long-term fatigue.

- **Creating a Balanced Routine with Leisure Activities**: Continuous work without leisure can drain motivation and creativity. Incorporating leisure activities into the daily routine provides a mental break and stimulates creativity. Dedicating time in the evening for hobbies, social activities, or simply relaxing with a book or movie can be refreshing. Balancing work with leisure activities promotes overall well-being and helps maintain motivation and creativity.

5.2.5: 5. NOTICE: How to See Clearly

Too often, we are physically with people but **not mentally present**. We struggle to notice them and see them truly. Being fully present makes others feel like the most important person in the world, even for a brief moment. This experience of undivided attention has a magical power that can leave a lasting impact long after the moment has passed.

People often describe the feeling of being with someone who is fully present as if that person had moved mountains for them. The power of presence is not about grand gestures but about being **wholly attentive** and **engaged** in the moment. This profound presence can transform relationships and create meaningful connections that resonate deeply.

Harness the power of presence to achieve a state of heightened awareness. Train your brain to focus on what is essential and ignore the irrelevant. This practice improves your productivity and enriches your interactions with others.

Set aside your opinions, advice, and judgment to truly see others. Prioritize their truth above your own. Clear the clutter in your physical environment before tackling the clutter in your mind. This process of decluttering helps create a space conducive to presence.

image by portra from istock

IT and software engineering professionals can achieve a state of heightened awareness, improving productivity and enriching their interactions with others. This approach fosters a work environment where presence lead to meaningful connections and effective collaboration.

Here are a few examples that IT architects as an inspiration to help their organizations see clearly:

- **Truly Seeing and Listening to Colleagues**: Meetings and discussions often involve multitasking, where participants check emails or think about other tasks, leading to ineffective communication. Active listening is essential to fully engaging with colleagues during meetings and one-on-one interactions. During a meeting, put away your phone and close unnecessary tabs on your computer. Make eye contact, listen actively, and acknowledge the speaker's points before responding. This engagement ensures that colleagues feel valued and heard, improving team dynamics and fostering a collaborative work environment.
- **Decluttering Physical and Digital Workspaces**: A cluttered workspace can lead to a cluttered mind, reducing efficiency and increasing stress. Clearing physical and digital clutter helps create an organized and focused work environment. An organized

workspace reduces distractions, helping you maintain focus and clarity throughout the day.

- **Prioritizing Important Tasks Over Urgent Ones**: Engineers often get caught up in urgent but less important tasks, neglecting critical long-term projects. Using a prioritization matrix like the Eisenhower Matrix helps distinguish between important and urgent tasks. Focusing on essential tasks enhances long-term productivity and progress, preventing the constant firefighting of urgent but less impactful tasks.

5.3: Effortless Action

When you reach an Effortless State, you can perform Effortless Actions. Effortless Action is the art of **accomplishing more by trying less**. It involves finding a natural flow in your tasks and responsibilities, allowing you to achieve your goals with minimal strain. The process begins with taking the first obvious step, which helps overcome procrastination and sets you in motion. By avoiding overthinking, you can focus on reaching completion without getting bogged down in unnecessary details, preventing mental fatigue and keeping you moving forward (Figure 3).

Progress in Effortless Action is made by **pacing yourself** rather than **powering through**. This sustainable approach ensures a steady momentum without the risk of burnout. Effortless Action allows you to exceed expectations without excessive effort, enabling you to overachieve while preserving your energy and well-being. Ultimately, it's about aligning your efforts with a natural rhythm, making work feel less like a struggle and more like a seamless part of your day.

***Figure 3**: Effortless Action is a part of the McKeown's Effortless framework (State, Action, Results).*

5.3.1: 6. DEFINE: What "Done" Looks Like

To begin an important project effectively, start by defining what "done" looks like. Establish **clear conditions for completion**, outlining specific

criteria that indicate the project is finished. This clarity **helps you stay focused** and prevents **unnecessary overextension**. Once these conditions are met, stop and acknowledge your progress.

McKeown suggests some simple techniques that everyone can use daily to focus and have a stronger sense of establishment. For instance, create a "**Done for the Day**" list, limited to items that would constitute meaningful progress. This list should include achievable tasks that contribute significantly toward your overall goal. Focusing on these critical activities ensures that each day ends with a sense of accomplishment and momentum.

image by evgenyatamanenko from istock

Similarly, by defining what "done" looks like, IT and software engineering professionals can stay focused on their goals, prevent overextension, and maintain a clear direction in their work. This approach fosters productivity, ensures high-quality outcomes, and provides a satisfying sense of accomplishment.

Here are a few examples that IT architects as an inspiration to help their organizations better define what "done" looks like:

- **Completing a Feature Development**: Developing a new feature

for a software application can often lead to scope creep without clear boundaries. To address this, defining what constitutes "done" for the feature is essential, as well as specifying criteria such as passing all unit and integration tests, undergoing code review, and updating documentation. By documenting these conditions—"Feature X is done when it passes all unit tests, integrates without errors, is reviewed by a peer, and is documented in the user manual"—the development team can stay focused, avoid unnecessary additions, and ensure timely completion.

- **Finishing a Bug Fix**: Bug fixes can create a cycle of finding new issues without a clear endpoint. To prevent this, determine the conditions under which the bug fix is complete. These conditions include reproducing the problem, implementing the fix, testing it in different environments, and updating the issue tracker. Document the steps required to complete the bug fix: "Bug Z is done when the issue is reproduced, fixed, tested in staging and production environments, and marked as resolved in the issue tracker." Explicit criteria for "done" help developers avoid endless bug-fixing cycles and ensure fixes are properly verified and documented.
- **Completing a Code Review**: Code reviews can drag on indefinitely without clear criteria for completion. To streamline this process, set clear criteria for a complete code review, such as checking for adherence to coding standards, verifying functionality, and ensuring no critical issues remain. Outline the steps: "The code review is done when the code adheres to our standards, all tests pass, and any identified issues have been addressed or noted for future improvement." Clear completion criteria make code reviews more efficient and ensure they add value without becoming a bottleneck.

5.3.2: 7. START: The First Obvious Action

"Done" not only helps finish a project, but it also helps start it. Establishing clear conditions for completion and outlining specific criteria that indicate the project is finished enables you to stay focused, prevents unnecessary overextension, and provides a clear starting point.

But "done" is not enough. People still get stuck because they do not know how to start. Make the **first action the most obvious one**. Break this

initial action into the tiniest, most concrete step possible, and then name it. For example, if the project is to write a report, the first step could be as simple as "open a new document."

McKeown suggests some simple techniques to identify the first obvious action:

- Spend **sixty seconds focusing** on your desired outcome. Visualize what success looks like, and remember this image as you work. This brief period of concentration aligns your efforts and provides clear direction, making it easier to start.
- Gain maximum learning from a minimal viable effort by starting with a **ten-minute microburst** of focused activity. This short, intense work period can boost motivation and energy, making diving deeper into the project more manageable. This approach helps you overcome the inertia of starting and builds momentum for continued progress.

image by christianchan from istock

By breaking down the first action into the most apparent, tiny steps and visualizing the desired outcome, IT and software engineering professionals can effectively begin essential projects, build momentum, and ensure clear direction from the start. This approach reduces inertia and makes it easier to dive deeper into tasks confidently.

Here are a few examples that IT architects as an inspiration to help their organizations find first obvious actions:

- **Beginning a New Feature Development**: In developing a new feature for a software application, the feature is considered complete when it passes unit tests, is integrated into the main codebase, and is documented. The first obvious action is to create a new branch in the version control system. This action involves opening the version control tool (e.g., Git) and executing the command to create a new branch: `git checkout -b new-feature-branch`. This small, concrete action provides a clear starting point and sets up the environment for feature development.
- **Initiating a Bug Fix**: When fixing a critical bug in the software, the bug fix is considered complete when the issue is reproduced, fixed, tested, and verified in the staging environment. The first obvious action is to reproduce the bug in the development environment. This action involves identifying the conditions under which the bug occurs and replicating those conditions in your development setup. Successfully reproducing the bug provides a clear starting point for identifying the cause and developing a fix.
- **Initiating a Documentation Task**: In updating the user manual with new feature details, the documentation update is complete when all new features are described with examples and integrated into the manual. The first obvious action is opening the text editor's documentation file. Locate the user manual file and open it using your preferred text editor (e.g., MS Word, Google Docs). Opening the document, while trivial, is an essential step in creating a starting point for adding new information.

5.3.3: 8. SIMPLIFY: Start With Zero

To simplify the process of completing an essential project, don't focus on simplifying the steps; instead, **remove unnecessary steps** altogether.

Recognize that not everything requires going the extra mile. Minimizing the actions you need to take will streamline your workflow and conserve energy. Maximize the steps not taken and measure progress in the smallest increments to ensure steady advancement.

This concept aligns with the "Swedish Death Cleaning philosophy," which involves decluttering your life by eliminating accumulated unnecessary items. Apply this principle to your project by removing redundant tasks and focusing only on what truly matters. This approach helps you maintain clarity and efficiency, ensuring that your efforts are directed toward meaningful progress without being bogged down by extraneous activities.

image by arismart from istock

By applying the "Start With Zero" approach, IT and software engineering professionals can eliminate unnecessary steps, streamline workflows, and focus on what truly matters. This approach enhances productivity and ensures that efforts are directed toward meaningful progress, making projects more efficient and manageable.

Here are a few examples that IT architects as an 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.

5.3.4: 9. PROGRESS: The Courage to Be Rubbish

When beginning a project, it is crucial to adopt the mindset that it is perfectly acceptable to **start with less-than-perfect work**. Embracing a "zero-draft" approach, simply putting any words on the page, can be incredibly liberating. This technique effectively bypasses the paralysis often caused by perfectionism, allowing creativity to flow more freely. Accepting the idea of failing cheaply and making small and manageable

mistakes early in the process accelerates learning and enhances decision-making skills over time.

Fred Brooks encapsulated this wisdom: *"Good judgment comes from experience, and experience comes from bad judgment."* This quote highlights the necessity of mistakes for achieving mastery. Initial drafts are not meant to be perfect. Courage to produce imperfect work is a vital component of growth. By starting messy and allowing for errors, a solid foundation for continuous improvement and eventual mastery is established.

image by cristian gheorghe from istock

By embracing the courage to be rubbish and starting with imperfect versions, IT and software engineering professionals can bypass perfectionism, accelerate learning, and create a foundation for continuous improvement and eventual mastery. This approach encourages experimentation and iteration, leading to better outcomes over time.

Here are a few examples that IT architects as an inspiration to help their organizations have the courage to create rubbish:

- **Starting a New Feature**: In developing a new feature for a software application, adopt the mindset that the first version might

be rudimentary and full of flaws. Begin with zero-draft code, writing the initial version of the feature without worrying about perfection. Focus on getting a basic version that works, even if it's not optimized or clean. Start by coding the main functionality with simple logic, ignoring optimizations for now. This "rubbish" version allows you to quickly identify the main challenges and requirements, setting a foundation for iterative improvements.

- **Initial Project Planning**: When planning a new software development project, accept that the first project plan will be incomplete and potentially unrealistic. Begin with a zero-draft plan, drafting a rough outline of the project timeline, key milestones, and resource allocations without striving for perfect accuracy. Create a simple Gantt chart or list of milestones using tools like Trello or a whiteboard. This initial plan provides a starting point for discussion and refinement, allowing the team to identify potential issues and adjust accordingly.
- **Learning a New Technology**: When learning a new programming language or framework, accept that your first attempts will be full of mistakes and inefficiencies. Begin with zero-draft learning, writing simple programs or small projects to get a feel for the syntax and features without worrying about best practices. Follow beginner tutorials and write code to replicate examples, making mistakes. These initial attempts build familiarity with the new technology and provide a foundation for more advanced learning and application.

5.3.5: 10. PACE: Slow if Smooth, Smooth is Fast

Maintaining a deliberate and **measured pace** can achieve more meaningful and lasting results without exhausting yourself. This balanced approach allows you to work efficiently while preserving your well-being and maintaining a high output standard.

To achieve sustained productivity, set an effortless pace: slow is smooth, smooth is fast. Reject the **false economy** of "**powering through**," which often leads to burnout and decreased efficiency. Instead, create a balanced approach to your work by defining a suitable range for your efforts: determine that you will never do less than X and never more than Y. This ensures a consistent, manageable workload that promotes steady progress.

Recognize that not all progress is created equal. Focus on the quality and significance of your achievements rather than merely the quantity.

image by flashvector from istock

By adopting a balanced approach and setting a sustainable pace, IT and software engineering professionals can maintain high productivity while ensuring quality and preserving their well-being. This method emphasizes the importance of consistent, manageable efforts that lead to meaningful and lasting results.

Here are a few examples that IT architects as an inspiration to help their organizations set an effortless pace:

- **Learning a New Technology**: When learning a new programming language or framework, the traditional approach might involve intensive, uninterrupted study sessions that lead to quick burnout and poor retention. A balanced approach involves studying for 30 minutes and no more than 2 hours per day. Integrate learning into daily routines with consistent, shorter study sessions, allowing time for reflection and practice. Steady, consistent learning leads to better understanding and retention of new skills.
- **Managing Code Development**: The traditional approach to developing a complex feature for a software application often involves powering through long hours of coding without breaks, leading to burnout and errors. A balanced approach involves determining that you will code for no less than 4 hours and 6 hours a day, with breaks

in between. Implement this by scheduling coding sessions with regular short breaks. This steady pace prevents burnout, reduces mistakes, and ensures high-quality code over time.

- **Debugging and Testing**: The traditional approach to debugging and testing a new feature often involves marathon sessions that lead to frustration and oversight. A balanced approach allocates no less than 1 hour and 3 hours daily for debugging and testing. Approach debugging methodically with regular breaks to reassess and strategize. This measured pace allows for more effective problem-solving and thorough testing, leading to more robust software.

5.4: Effortless Results

Effortless Results are the natural outcome of consistently cultivating your Effortless State and taking Effortless Action. By maintaining a clear objective, breaking tasks into tiny, obvious first steps, and working at a consistent, manageable pace, you achieve your desired outcomes with greater ease. However, the true power of Effortless Results lies in their sustainability (Figure 4).

Effortless Results are those that continue to **flow to you repeatedly**, with minimal additional effort. You've established a system where **success becomes a cycle** rather than a one-time event. By refining your process and eliminating unnecessary steps, you ensure that your efforts yield ongoing benefits. This approach allows you to maintain high productivity and achieve your goals consistently, creating a seamless and continuous flow of accomplishments.

In essence, Effortless Results are about creating a **self-sustaining loop** of success. By leveraging the principles of the Effortless State and Effortless Action, you set the stage for ongoing achievement, making it possible to reach your objectives repeatedly without constant exertion.

Figure 4: *Effortless Results is a part of the McKeown's Effortless framework (State, Action, Results).*

5.4.1: 11. LEARN: Leverage the Best of What Others Know

To achieve Effortless Results, McKeown argues, it is crucial to **stand on the shoulders of giants**. That is, to leverage the best of what experts and pioneers have discovered. Use their knowledge as a foundation to build upon, enabling you to achieve more with less effort.

In addition, one must focus on **learning principles**, not just facts and methods. Understanding first principles allows you to apply them repeatedly across various contexts. You can quickly adapt and innovate by grounding yourself in these fundamental truths, making complex tasks more straightforward and manageable.

Effortless Results stem from a **deep understanding** of **first principles** and the ability to apply them creatively and consistently. By building on the knowledge of others and developing your unique insights, you create a sustainable pathway to continuous achievement and innovation.

image by lvcandy from istock

By understanding and applying first principles, leveraging experts' knowledge, and building unique insights, IT and software engineering professionals can achieve effortless results. This approach fosters innovation and continuous improvement, ensuring that complex tasks become more manageable and maximize productive efforts.

Here are a few examples that IT architects as an inspiration to help their organizations leverage the best of what others know:

- **Applying Design Patterns in Software Development**: When developing a scalable and maintainable software application, start with the first principles by learning the fundamentals behind common design patterns such as Singleton, Factory, Observer, and MVC. Study books like "Design Patterns: Elements of Reusable Object-Oriented Software" by the Gang of Four. Apply these

patterns appropriately in the software architecture to address scalability and maintainability issues. Leveraging well-established design patterns allows for efficiently building a robust, proven, and flexible application framework.

- **Innovating with New Technologies**: To integrate machine learning into an existing product, start with the first principles by understanding the basics of machine learning algorithms, data preprocessing, model training, and evaluation. Use open-source libraries like TensorFlow or PyTorch and build on existing models and research to implement the solution. Using foundational knowledge and the work of experts facilitates the seamless integration of advanced technologies, enhancing functionality with minimal effort.
- **Optimizing Database Performance**: To optimize the performance of a relational database, start with first principles by learning the fundamentals of database normalization, indexing, query optimization, and transaction management. Study best practices and methodologies from experts. Apply indexing strategies, optimize queries based on execution plans, and configure database settings according to expert recommendations. Achieving optimal database performance through a deep understanding of underlying principles and expert advice leads to faster and more efficient data handling.

5.4.2: 12. LIFT: Harness the Strength of Ten

Teaching others is an accelerated way to learn. When you prepare to teach, you increase your engagement, focus more intently, listen to understand, and think about the underlying logic so you can articulate the ideas in your own words. This process reinforces your understanding and enhances your ability to convey complex concepts.

Use teaching as a lever to harness the strength of ten, achieving a **far-reaching impact** by teaching and by **teaching others to teach**. You'll notice how much you learn when you live what you teach. You can disseminate knowledge effectively by telling stories that are easy to understand and repeat.

Ensuring your messages are easy to understand and hard to misunderstand is not just a communication strategy. It's a powerful tool for **amplifying your impact**. By simplifying and clarifying your communication,

you make it easier for others to grasp and remember the knowledge you share, thereby increasing the reach and effectiveness of your message. This process underscores your crucial role in knowledge sharing and the impact you can have on others.

For instance, one motivation for writing this book is to create reusable material that teaches others about pragmatic approaches to running an architecture practice. It also gives them material they can reuse to teach others the same.

image by nattakorn maneerat from istock

By teaching and teaching others to teach, IT and software engineering professionals can amplify their impact and foster a culture of continuous learning and improvement. This approach reinforces their knowledge and equips the team with the skills and understanding necessary for greater efficiency and innovation.

Here are a few examples that IT architects as an inspiration to help their organizations harness the strength of ten:

- **Promoting Continuous Integration/Continuous Deployment (CI/CD)**: An engineer who has successfully implemented CI/CD

pipelines in past projects runs workshops to teach the team how to set up and maintain CI/CD pipelines. Providing hands-on experience with popular tools like Jenkins, GitHub Actions, or GitLab CI, the engineer creates step-by-step tutorials and shares best practices. Using real examples from the current project to demonstrate the impact of CI/CD, the team learns to automate testing and deployment, leading to more reliable releases and a faster development cycle.

- **Enhancing Security Practices**: A security expert aims to improve the team's approach to cybersecurity by conducting security training sessions to educate team members on best practices, common vulnerabilities, and mitigation strategies. Using simple, memorable stories to illustrate the importance of security, the expert provides checklists and templates for secure coding and threat modeling. The outcome is that team members become more security-conscious and capable of implementing robust security measures, reducing the risk of breaches.
- **Mentoring on Effective Documentation**: A developer who excels at creating clear, concise documentation holds workshops to teach the team how to write effective documentation. Sessions cover different types of documentation, such as ADRs, RfCs, API documentation, user guides, and technical specs. By providing examples of well-written documentation and highlighting key elements that make it effective, the developer encourages team members to practice and peer-review each other's work. Improved documentation quality across the team leads to better knowledge sharing and easier onboarding for new team members.

5.4.3: 13. AUTOMATE: Do It Once and Never Again

Automating as many essential tasks as possible frees up space in your brain. This space allows you to focus your mental energy on more important matters. One effective way to ensure consistency and accuracy is using checklists, which help you get it right every time without relying on memory. Automation creates a more efficient and less cluttered mental environment, enabling you to perform at your best with minimal effort.

image by shutter2u from istock

By automating essential tasks and using checklists for routine processes, IT and software engineering professionals can streamline their workflows, reduce errors, and free up mental space for more important activities. This approach leads to increased efficiency, consistency, and overall productivity.

Here are a few examples that IT architects as an inspiration to help their organizations automate things:

- **Automating Code Quality Checks**: Ensuring code quality through manual reviews and testing can be time-consuming and inconsistent. A better approach involves high-tech automation: Integrate automated code quality tools like SonarQube or ESLint into your CI/CD pipeline. Configure the tools to run automated checks on every commit, providing immediate feedback on code quality issues. The outcome is consistent and automated code quality checks that improve overall code standards and reduce the need for manual reviews.
- **Standardizing Development Environments**: Setting up development environments manually for new team members or new projects can be time-consuming and error-prone. The approach involves high-tech automation by using containerization tools like Docker to create standardized development environments. Create

Docker images that include all necessary tools, libraries, and 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.

5.4.4: 14. TRUST: The Engine of High-Leverage Teams

When trust exists in your relationships, they require less effort to maintain and manage. You can quickly and efficiently split work between team members. People feel comfortable **discussing problems openly and honestly**, sharing valuable information rather than hoarding it. This environment encourages team members to ask questions when they don't understand something. Consequently, the speed and quality of decisions improve, political infighting decreases, and you may even enjoy the experience of working together. This dynamic allows you to focus your energy and attention on getting important tasks done rather than on simply getting along.

Conversely, when trust is low, **everything becomes difficult**. Sending a simple text or email is exhausting as you weigh every word for how it might be perceived. Responses may induce anxiety, and every conversation feels like a grind. Without trust in someone's ability to deliver, you need to check up on them constantly, remind them of deadlines, hover over their work, or avoid delegating tasks altogether, believing it's easier to do it yourself. This lack of trust can cause work to stall and impede team performance.

Inside every team are individuals with interrelated roles and responsibilities, moving at high speeds. Without trust, conflicting goals, priorities,

and agendas create friction and wear everyone down. **Trust acts like engine oil**, lubricating the team's interactions and keeping them working smoothly together. A team running out of trust will likely stall or sputter out.

Every relationship involves three parties, **Person A** and **Person B**, and **the structure** that governs them. When trust becomes an issue, most people point fingers at the other person, be it the manager blaming the employee or vice versa. However, every relationship has a structure, even if it's an unspoken and unclear one. Unclear expectations, incompatible or conflicting goals, ambiguous roles and rules, and misaligned priorities and incentives characterize a low-trust structure.

High-trust agreements help mitigate these issues by clarifying:

- **Results**: What results do we want?
- **Roles**: Who is doing what?
- **Rules**: What minimum viable standards must be kept?
- **Resources**: What resources (people, money, tools) are available and needed?
- **Rewards**: How will Progress be evaluated and rewarded?

Establishing clear expectations and structures creates a high-trust environment that enables teams to function efficiently and effectively.

Leverage trust as the essential lubricant of frictionless and high-functioning teams. Making the right hire once can produce results repeatedly. Follow the Three I's Rule: hire people with integrity, intelligence, and initiative. Design high-trust agreements to clarify results, roles, rules, resources, and rewards.

Being an architect is much **easier in high-trust organizations**. In low-trust organizations, people frequently expect architects to be **police agents**. IT governance processes frequently associated with an architecture practice are used or misused to force bureaucratic controls as teams often do not trust each other.

image by nathaphat from istock

By establishing and maintaining a high-trust environment, IT and software engineering teams can work more efficiently and effectively. Trust reduces friction, enables **better decision-making**, and fosters a positive, collaborative culture where team members feel valued and empowered.

Here are a few examples that IT architects as an inspiration to help their organizations create a high-trust environment:

- **Open Communication and Problem-Solving**: To address a critical bug in the software that could delay the release, encourage an environment where team members feel comfortable discussing problems openly. Regularly schedule team stand-ups and retrospectives where issues can be raised without fear of blame. Encourage a culture where questions are welcomed, and information is shared freely. The outcome is an effective collaboration among team members to identify and resolve the bug quickly, leveraging their collective knowledge and skills.
- **Delegating Tasks Efficiently**: When delegating a complex module development to a junior engineer or IT architect, trust their capability to handle the task while providing guidance and support. Communicate the task requirements and expected outcomes. Provide access to resources and be available for consultation, but allow the engineer autonomy to approach the problem. The result is that the junior engineer gains confidence and develops skills, while senior team members can focus on other critical tasks, improving overall team productivity.

- **Making the Right Hire**: When hiring a new software engineer for a critical project, the approach involves following the Three I's Rule: focus on candidates with integrity, intelligence, and initiative. Design interview questions and assessments that evaluate these traits, such as scenario-based questions to gauge problem-solving skills and ethical dilemmas to assess integrity. The outcome is hiring the right candidate who increases team efficiency and reduces the need for constant oversight, as they can be trusted to deliver high-quality work independently.

5.4.5: 15. PREVENT: Solve the Problem Before It Happens

Just as you can find small actions to make your life easier in the future, look for small actions that will prevent your life from becoming more complicated. Simple preventative measures (such as setting reminders, automating tasks, or creating checklists) can significantly reduce the likelihood of future problems. Focusing on these small, strategic actions ensures a smoother, more efficient path forward, allowing you to achieve more with less effort and stress.

In other words, don't just manage problems—solve them before they happen. Seek simple actions today that can prevent complications tomorrow. Finding opportunities to invest a small amount of effort now can eliminate recurring frustrations and streamline your future.

This proactive approach is the long tail of time management. When you invest your time in actions with a long tail, you reap the benefits over a long period. For example, spending two minutes to organize your workspace can save you countless hours of searching for items in the future. Catch mistakes before they occur by adopting a measure-twice, cut-once mentality.

image by ipuwadol from istock

IT and software engineering teams can solve potential problems before they occur, leading to smoother operations, fewer disruptions, and a more efficient workflow. This proactive approach saves time and resources, allowing teams to focus on innovation and continuous improvement.

Here are a few examples that IT architects as an inspiration to help their organizations prevent problems:

- **Regular Code Reviews**: Establishing a regular code review process can prevent inconsistent code quality and reduce the number of errors from unreviewed code changes. Make code reviews a mandatory part of the development workflow. Use pull request templates and review tools like GitHub or GitLab to ensure thorough reviews. The outcome is that regular code reviews maintain high code quality, catch potential issues early, and promote knowledge sharing among team members.
- **Setting Up Monitoring and Alerts**: Implementing monitoring and alerting systems can prevent the frequent discovery of system outages or performance issues after they impact users. Use tools like Prometheus, Grafana, and New Relic to monitor system performance and uptime. Set up alerts to notify the team of any anomalies or performance degradation. The outcome is early detection of

issues, allowing the team to address problems before they impact users, improving system reliability and user satisfaction.

- **Regular Security Audits**: Discovering security vulnerabilities after a breach can lead to significant downtime and data loss. You can reduce such problems with preventative actions, such as conducting regular security audits and implementing best practices. Schedule regular security audits using tools like OWASP ZAP and Nessus. Train the team on secure coding practices and conduct periodic security reviews. The outcome is that regular security audits and training reduce the risk of vulnerabilities, preventing potential breaches and ensuring data protection.

5.5: The Road to Effortless Achievement

The path to achieving great results doesn't have to be tortuous. Embracing the idea that things can be easy and even enjoyable opens you up to effortless solutions. By clearly defining your goal and identifying a small yet significant first step, you set yourself on a journey of effortless actions that lead to steady and meaningful progress.

When you leverage knowledge, automation, and trust, you create a system that produces recurring results. This system simplifies your life and enhances productivity, demonstrating that life doesn't have to be as hard and complicated as we often make it. Adopting a mindset that prioritizes ease and enjoyment enables a more streamlined and fulfilling approach to achieving your organizational goals.

5.6: To Probe Further

- Effortless: Make It Easier to Do What Matters Most[4], by Greg McKeown, 2021
- Essentialism: The Disciplined Pursuit of Less[5], by Greg McKeown, 2020
- Here's why enterprise IT is so complex[6], by Gregor Hohpe, 2018
- Tidy First?[7], by Kent Beck, 2023
- Execution Management Matrix[8]
- The Eisenhower Matrix[9]

[4] https://gregmckeown.com/books/effortless/
[5] https://gregmckeown.com/books/essentialism/
[6] https://architectelevator.com/architecture/it-complexity/
[7] https://www.oreilly.com/library/view/tidy-first/9781098151232/
[8] https://obren.io/tools/matrix/?matrix=executionManagement
[9] https://obren.io/tools/matrix/?matrix=eisenhowerMatrix

5.7: Questions to Consider

- *How does the idea of achieving goals with minimal strain resonate with your personal work ethic and habits?*
- *What tasks could benefit from automation to enhance your productivity and reduce unnecessary effort?*
- *Have you experienced moments when you were physically rested, emotionally unburdened, and mentally energized? What facilitated these moments?*
- *What mental clutter currently hinders your productivity, and how can you clear it to foster an effortless state?*
- *How can you apply the concept of "make the change easy, then make the easy change" in your work?*
- *How can you combine essential tasks with pleasurable activities to enhance your productivity and well-being?*
- *What regrets, grudges, or unrealistic expectations are you holding onto that might be hindering your progress?*
- *How do you currently balance work and rest to prevent burnout and maintain high performance?*
- *What structured routines can you implement to incorporate rest and reflection periods into your workday?*
- *What steps can you take to declutter your physical and digital workspaces to improve focus and efficiency?*
- *What experts and pioneers in your field can you learn from to build on their knowledge and achieve more with less effort?*
- *How can you leverage teaching as a tool to reinforce your knowledge and multiply your impact?*
- *What complex concepts can you simplify and clarify to make them easier for others to understand and remember?*
- *What clear expectations and structures can you establish to create a high-trust agreement in your projects?*
- *How can you adopt a proactive approach to identify and solve potential problems before they occur?*

Part II: On Strategy

6: On Strategy

image by eoneren from istock

IN THIS SECTION, YOU WILL: Get a summary of several resources on strategy that I use as inspiration for running the Grounded Architecture practice in complex organizations.

There are many definitions of business strategy, but I found the view of Roger L. Martin[1], a prominent business strategist, the most practical and inspiring. Maring views strategy as integrated choices designed to achieve a specific objective. He emphasizes that strategy is not merely a plan but a framework or theory for making decisions that guide an organization toward success.

In his work, Martin introduces the "Strategy Choice Cascade," which outlines five critical questions that form the foundation of a robust strategy:

1. **What is our winning aspiration?**
2. **Where will we play?**
3. **How will we win?**
4. **What capabilities must be in place to win?**
5. **What management systems are required to ensure the capabilities are in place?**

This cascade encourages organizations to make deliberate choices that align with their overarching goals, ensuring that each decision reinforces the others to create a cohesive strategy.

Roger L. Martin emphasizes the importance of articulating the underlying logic of a strategy. He suggests that strategists clearly define the cause-and-effect relationships they believe will lead to success. This definition involves specifying the assumptions and conditions that must hold true for the strategy to be effective. By laying out this logic, organizations can better understand their strategic choices and identify potential areas of risk or uncertainty. Martin advises that after establishing this logical framework, organizations should implement the strategy, monitor its progress, and make adjustments as necessary. This approach acknowledges the inherent uncertainties in strategic decision-making and promotes adaptability in changing circumstances.

Business strategy and IT are deeply intertwined, with **business processes** and **supporting systems** working together to achieve organizational goals. Strategy sets the direction, identifying goals, priorities, and specific needs, while IT enables execution by streamlining processes, driving innovation, and providing data-driven insights for decision-making.

[1] https://rogerlmartin.com/thought-pillars/strategy

This relationship is dynamic, where business needs shape IT solutions, and technological advancements influence strategic opportunities. New technology also requires changing how you manage it, as traditional management approaches may not suffice for emerging technologies' unique demands. Effective alignment requires integrating processes with systems, adapting workflows to leverage technology, and fostering collaboration between business leaders and IT teams. As both evolve, continuous realignment ensures organizations remain agile and competitive, maximizing the mutual value of business and IT.

In business strategy and IT architecture, I often draw inspiration from several key sources, covered in the follow-up sections:

- **Enterprise Architecture as Strategy**[2]: The relationship between business strategy and IT architecture is dynamic and requires continuous realignment, leveraging different operating models, flexibility strategies, and IT maturity stages to maximize efficiency, agility, and strategic value.
- **Outsourcing Strategies**[3]: Outsourcing strategies—strategic outsourcing, co-sourcing, and transactional outsourcing—help organizations optimize external partnerships based on business goals, operational needs, and desired control levels, with IT architecture playing a critical role in ensuring integration, governance, and alignment with enterprise objectives.

In the Appendix, I also put my notes on diverse strategy models:

- **Achieving Market Leadership**[4]: The Discipline of Market Leader framework highlights three distinct strategies for market leadership—operational excellence, product leadership, and customer intimacy—each requiring a tailored IT architecture to align technology with business goals, optimize processes, and sustain a competitive edge.
- **Value-Based Strategy**[5]: An IT architecture can align technology decisions with value-based business strategy by simplifying IT initiatives, enhancing customer experience, improving employee

[2] ea-as-strategy
[3] outsourcing
[4] market-leadership
[5] value-based-strategy

work conditions, optimizing supplier relationships, and enabling business model shifts to maximize overall value creation.

- **Connecting Marketing, Sales, and Customer Service Strategies**[6]: IT architects play a vital role in optimizing marketing, sales, and customer care by integrating technology, data, automation, and security to create seamless customer experiences, improve efficiency, and drive business growth.
- **Culture As a Strategy (aka Culture Eats Strategy for Breakfast)**[7]: Organizational culture profoundly influences IT architecture by shaping how architects approach collaboration, decision-making, innovation, and technical priorities, requiring alignment between cultural values and architectural strategies for long-term success.

[6] marketing-sales-strategy
[7] culture-strategy

7: Enterprise Architecture as Strategy

image by chunyip wong from istock

IN THIS SECTION, YOU WILL: Learn that the relationship between business strategy and IT architecture is dynamic and requires continuous realignment, leveraging different operating models, flexibility strategies, and IT maturity stages to maximize efficiency, agility, and strategic value.

KEY POINTS:

- Business Operating Models: Organizations can structure their IT and business processes around four models—Diversification, Coordination, Replication, and Unification—depending on their need for standardization and integration.
- Global vs. Local Flexibility: Companies must balance global efficiency with local adaptability by implementing modular architectures and governance structures that support both.
- Stages of IT and Strategy Alignment: Organizations evolve from business silos (fragmented systems) to business modularity (strategic agility), with increasing levels of IT-business integration.
- IT's Strategic Role: IT architecture serves as a key enabler for business execution, operational efficiency, and innovation, requiring alignment with the company's operating model and strategic goals.
- Evolving IT Maturity: Progressing through IT maturity stages demands investment in technology, governance, and cultural shifts, ensuring seamless integration between business and IT functions.

The relationship between IT and business is **inherently dynamic**, where **business needs shape IT solutions**, and **technological advancements influence strategic opportunities**. New technology also demands **a shift in management approaches**, as traditional methods may not meet the **unique demands of emerging technologies**. Effective alignment requires **integrating processes with systems**, **adapting workflows to leverage technology**, and fostering collaboration between business leaders and IT teams. As both domains evolve, **continuous realignment ensures agility**, helping organizations remain competitive and maximize the **mutual value of business and IT**.

In business strategy and IT architecture, I often draw inspiration from *Enterprise Architecture as Strategy: Creating a Foundation for Business Execution*[1] by Jeanne W. Ross, Peter Weill, and David C. Robertson.

[1] https://store.hbr.org/product/enterprise-architecture-as-strategy-creating-a-foundation-for-business-execution/8398

I apply several **core concepts from these works** to shape my understanding of the **interplay between business strategy and IT architecture**:

- **Business Operating Models** define how an organization structures its processes and systems to deliver value. These models include ***Diversification***, where units operate autonomously in distinct markets; ***Coordination***, where units share data and resources to enhance collaboration; ***Replication***, with independent units using standardized systems; and ***Unification***, where **tightly integrated processes** ensure **efficiency and strategic alignment**.
- **Global vs. Local Business Flexibility** highlights the **tension between centralized control and decentralized agility**. ***Global flexibility*** enables enterprise-wide responsiveness to change, while ***local flexibility*** empowers units to **address specific regional or functional needs**. Striking the right balance ensures **adaptability without fragmentation**.
- **Stages of IT and Strategy Alignment** describe the **evolution of IT's role in strategic execution**, moving from isolated systems to **fully integrated, strategic platforms**. These stages include ***business silos*** (low integration), ***standardized technology*** (shared platforms, fragmented processes), ***optimized core*** (aligned and standardized systems), and ***business modularity***, which promotes **innovation through reusable, flexible components**.

Together, these concepts form a **comprehensive framework** for understanding how **IT architecture enables strategic business execution** and drives **enterprise-level transformation**.

7.1: Four Types of Business Operating Models

The **Operating Model** is a core concept in *Enterprise Architecture as Strategy*, representing how an organization **structures its business processes and systems to deliver value**. It determines the degree of business process **standardization (consistency)** and **integration (information sharing)** across business units. The book introduces the concept of the Operating Model, which defines the necessary level of business process standardization and integration.

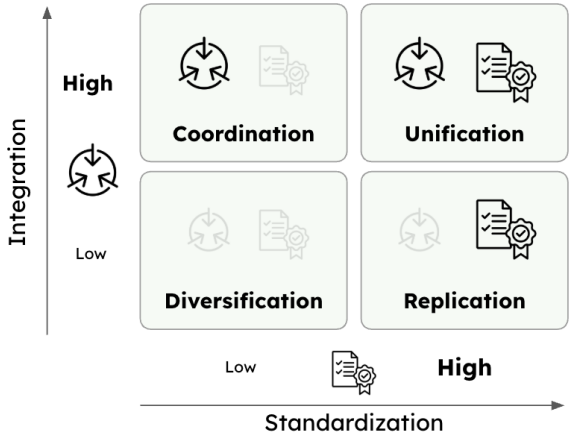

***Figure 1**: Four types of operating models: diversification (low integration, low standardization), coordination (high integration, low standardization), replication (low integration, high standardization), and unification (high integration, high standardization).*

There are four types of operating models (Figure 1):

- **Diversification**: Low integration, low standardization.
- **Coordination**: High integration, low standardization.
- **Replication**: Low integration, high standardization.
- **Unification**: High integration, high standardization.

7.1.1: Diversification Model

The **Diversification Model** features **low integration** and **low standardization**, allowing business units to operate independently. This approach targets **specific markets** or customer segments with **minimal need for sharing data or processes**.

image by elenabs from istock

Pros:

- **Flexibility and Innovation**: The model promotes agility at the unit level, facilitating quick responses to market-specific demands.
- **Localized Focus**: Each unit can tailor its strategies and operations to effectively serve its unique markets or products.
- **Autonomy**: Business units prioritize independence, which can lead to specialized expertise and localized innovation.

Cons:

- **Missed Synergies**: The independent operation of business units often results in missed opportunities for cost savings and collaboration.
- **Economies of Scale**: Difficulty in achieving economies of scale due to the lack of integration among units.

- **Increased Complexity**: If integration becomes necessary later, it may lead to challenges given the autonomous nature of the business units.

This model is particularly suitable for **holding companies** or **conglomerates**, where **operational overlap is limited** and strategic focus varies among units.

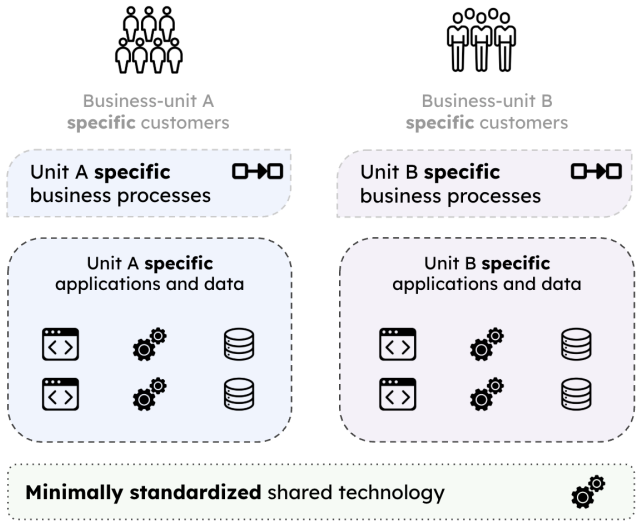

***Figure 2**: A high-level core diagram of the diversification operating model.*

In the **Diversification Operating Model**, where **low integration and low standardization** are key characteristics, the **IT architecture** and the **work of IT architects** must align with the model's emphasis on autonomy, flexibility, and localized innovation (Figure 2). Here's a typical breakdown:

Federated or Decentralized IT Architecture

- Each business unit (BU) maintains its **own IT systems**, infrastructure, and applications.

- There is **no centralized IT backbone**; instead, units choose solutions that best fit their specific needs.
- Integration between units is minimal or non-existent.

Independent Technology Stacks

- Different BUs may use different vendors, platforms (e.g., AWS in one BU, Azure in another), and ERP systems.
- Custom-built or best-of-breed systems tailored to the unit's industry or market.

Minimal Shared Services

- Little to no shared databases, CRM, or ERP systems.
- Exceptions might be for corporate-level compliance or financial reporting systems.

Focus on API Gateways or Minimal Interfacing

- If any communication is needed between units or with headquarters, lightweight APIs or integration layers may be used.

High Emphasis on Security and Governance at Unit Level

- Security is managed locally with standards possibly guided but not enforced by corporate IT.

Solution Architecture at the BU Level

- Architects work within individual business units, focusing on **designing localized solutions** that align with BU goals.
- Heavy involvement in **vendor selection**, customization, and integration within the unit.

Enterprise Architecture (EA) Plays a Loose Governance Role

- EA provides **guiding principles** but does not enforce strict alignment across units.

- May define **broad security, compliance, and interoperability standards** if needed.

Focus on Enabling Agility and Autonomy

- Architects prioritize fast deployment cycles, agile delivery methods, and flexibility in tech choice.
- Responsible for **building architecture that supports fast innovation** and rapid response to market needs.

Limited Involvement in Cross-Unit Standardization

- Minimal focus on system consolidation or enterprise-wide standardization.
- Instead, architects help manage **technical debt** and **complexity** within their own units.

Strategic Evaluation for Integration Opportunities (If Needed)

- Occasionally, architects may evaluate **potential integration points** (e.g., common analytics layer) to balance autonomy with potential synergies.

Aspect	Characteristic
IT Architecture Style	Decentralized / Federated
Technology Choices	Independent per BU
Integration Level	Minimal
Data Sharing	Very limited or none
IT Architect Focus	BU-specific solutions, flexibility, innovation
EA Role	Loose oversight, optional guidelines
Key Challenges	Managing complexity, avoiding duplicated efforts, ensuring security compliance

7.1.2: Coordination Model

The **Coordination Model** is characterized by **high integration** and **low standardization**, allowing business units to operate independently while collaborating through shared data and resources (Figure 3). This approach is particularly effective in organizations with distinct units, such as **healthcare systems**, where hospitals, clinics, and pharmacies work together using **shared patient records**. It is also applicable in **global supply chains**, where **inventory and demand data** are exchanged across geographically dispersed warehouses and suppliers.

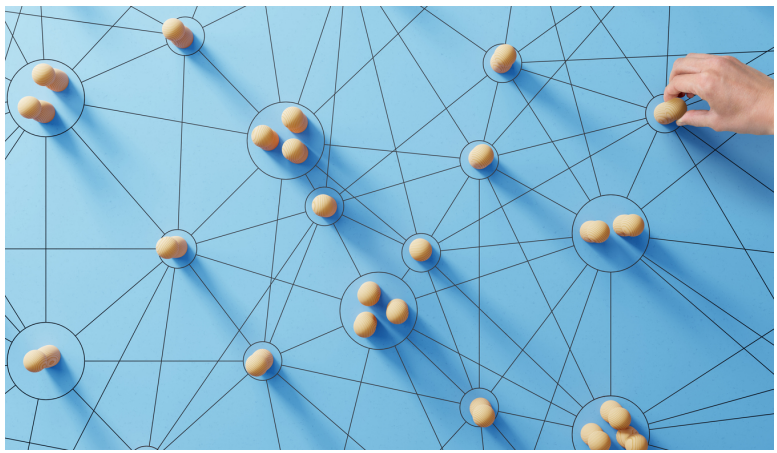

image by nicoelnino from istock

Pros:

1. **Enhanced Collaboration**: The model prioritizes collaboration, facilitating efficient access and exchange of critical information across different parts of the organization.
2. **Informed Decision-Making**: By enabling access to shared data, it supports informed decision-making, allowing business units to leverage insights from various sources.
3. **Fosters Innovation**: The emphasis on collaboration encourages innovation at the individual business unit level without enforcing rigid standardization, allowing for tailored approaches.

4. **Flexibility**: Business units can operate independently, adapting their processes to meet specific needs while still benefiting from shared resources.

Cons:

1. **Complexity of Integration**: Managing the integration of diverse systems and processes can be challenging, particularly when dealing with varied technologies and data formats.
2. **Dependence on Technology**: The model requires robust integration technologies and governance mechanisms to maintain cohesion, which can lead to increased costs and resource allocation.
3. **Governance Challenges**: Establishing effective governance structures to oversee data sharing and integration can be complex and may require continuous adjustments.

In summary, while the **Coordination Model** offers significant advantages in enabling collaboration and innovation, it also poses challenges related to system integration and governance that organizations must address to ensure successful implementation.

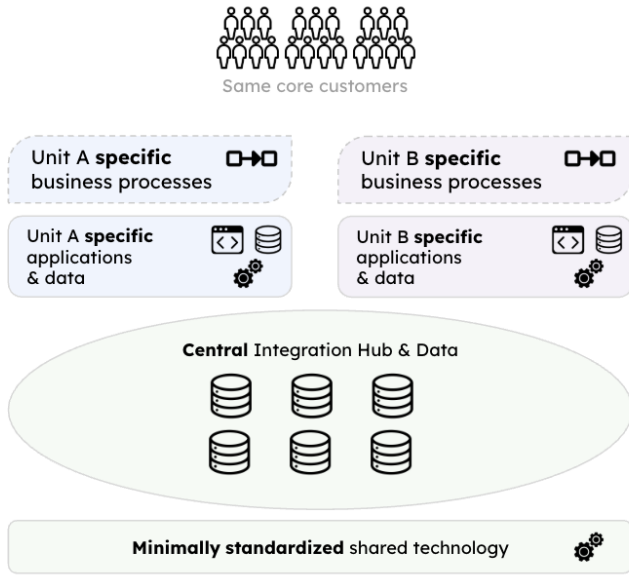

***Figure 3**: A high-level core diagram of the coordination operating model.*

The **Coordination Operating Model**, which emphasizes **high integration** and **low standardization** across independent business units, leads to several requirements for IT architecture and the work of IT architects:

Federated Architecture with Strong Integration Backbone

- Business units maintain **independent applications and processes** but **share key data** through integration platforms (e.g., patient records, inventory data).
- While there is **no enforced process standardization**, **data consistency and availability** across units are essential.

Integration-Centric Infrastructure

- Centralized or distributed **integration platforms** (e.g., ESB, API gateways, iPaaS) facilitate secure, real-time data exchange.
- There is a focus on **middleware and interoperability standards** (e.g., HL7 in healthcare, EDI in supply chains).

Shared Data Repositories

- Shared master data (e.g., customer, patient, product, supplier records) require strong **data governance and access controls**.
- The use of **data lakes or federated databases** supports analytics and cross-unit insights.

Domain-Oriented Architecture (e.g., Data Mesh Principles)

- Units are responsible for their own data domains but publish data to a shared infrastructure for enterprise-wide use.
- This approach promotes **decentralized ownership** along with **central coordination**.

Secure, Scalable Network and Identity Management

- **Federated identity and access management (IAM)** systems enable secure data access across units.
- Robust **data encryption**, **audit trails**, **and access policies** are essential.

Integration Architecture Leadership

- IT architects design and oversee **data integration strategies**, ensuring consistency, security, and performance across all units.
- They are responsible for the selection and implementation of **integration technologies**, protocols, and standards (e.g., RESTful APIs, GraphQL, Kafka, HL7).

Data Governance and Architecture

- Architects assist in defining **shared data models**, master data strategies, and governance frameworks.
- They work closely with data stewards and governance bodies to ensure **data quality and lifecycle management**.

Cross-Domain Collaboration

- Facilitate coordination between independent business units to establish **interoperability requirements** and **data-sharing agreements**.
- Help balance local autonomy with the need for **enterprise-wide insights**.

Enterprise Architecture as an Enabler

- Enterprise Architecture (EA) functions as a **collaborative enabler** rather than a central authority, setting guidelines for integration and shared platforms.
- There is a focus on **capability mapping**, **integration blueprints**, and **reference architectures**.

Strategic Innovation Support

- Enable experimentation and innovation within business units while ensuring alignment with **shared infrastructure and data strategies**.
- Architect solutions that support **future scalability** and the **evolution of business needs**.

Aspect	Characteristic
IT Architecture Style	Federated with shared integration and data platforms
Technology Choices	BU-specific systems, but standardized integration
Integration Level	High – focused on data sharing and interoperability
Data Sharing	Extensive across units (e.g., patient records, supply chain data)
IT Architect Focus	Integration, data architecture, governance, enabling autonomy with coordination
EA Role	Collaborative enabler; sets integration/data standards without enforcing process uniformity

Aspect	Characteristic
Key Challenges	Data consistency, integration complexity, governance at scale

7.1.3: Replication Model

The **Replication Model** is characterized by **low integration** and **high standardization**, enabling business units to operate independently while adhering to **standardized systems and processes** (Figure 4). This model is particularly effective for **franchise operations**, such as **McDonald's** and **Starbucks**, and for **hotel chains**, where each location maintains **operational autonomy** but follows **uniform guidelines** for service and operations.

image by bim from istock

Pros:

- **Scalability**: Facilitates rapid expansion by allowing proven operations to be easily replicated across multiple locations.

- **Consistency**: Ensures a uniform customer experience and operational standards, which is vital in brand-driven industries.
- **Efficiency of Proven Systems**: By utilizing established processes and systems, companies can reduce the learning curve for new locations.

Cons:

- **Limited Information Sharing**: The independence of business units can hinder collaboration and the flow of valuable insights among locations.
- **Potential Inefficiencies**: Local units may face challenges if unique processes are required, which can complicate the standardized approach and lead to operational inefficiencies.

Overall, the **Replication Operating Model** effectively balances operational autonomy with consistency, enabling businesses to scale swiftly while maintaining their brand's integrity. However, it is essential to be aware of its limitations to ensure that operational efficiency is not compromised.

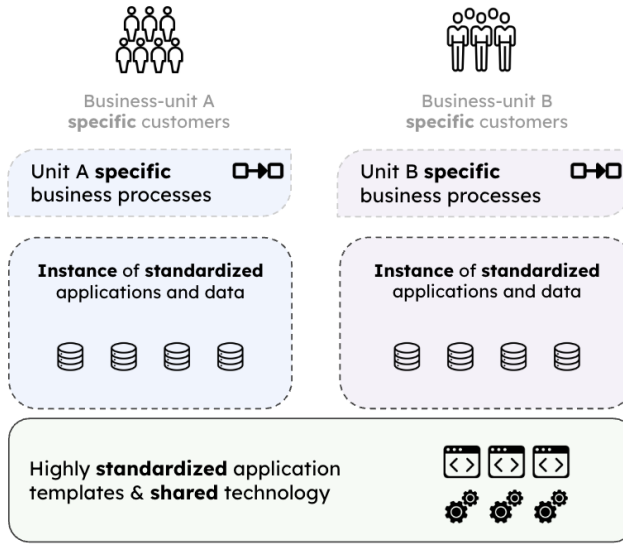

***Figure 4**: A high-level core diagram of the replication operating model.*

Here is a structured overview of the typical **IT architecture** and **IT architect responsibilities** for the **Replication Operating Model**. This model features **low integration** and **high standardization**, making it ideal for franchises and operations that require global consistency:

Centralized Standards, Decentralized Execution

- **Standardized applications and infrastructure** are deployed across all business units or locations.
- Each unit operates its own instance or environment while adhering to **uniform processes, templates, and configurations**.

Uniform IT Stack Across Units

- Common enterprise systems (e.g., POS, CRM, HR, inventory) are utilized globally with an **identical setup**.
- Cloud or hybrid models often facilitate rapid deployment and centralized updates.

Minimal Real-Time Data Integration

- Units do **not share operational data** in real time with each other.
- Periodic data aggregation may occur at a central location for reporting, compliance, or corporate oversight.

Standardized Deployment and Support Models

- **Infrastructure-as-Code (IaC)** and automation are employed to replicate environments quickly and consistently.
- Central IT may provide updates, patches, and support through a shared services model.

Controlled Autonomy

- Business units have **limited flexibility** in customizing systems to ensure **global uniformity**.
- Central governance enforces compliance with established standards.

Platform and Application Standardization

- Architects design **uniform IT solutions** that can be deployed across units with minimal customization.
- There is an emphasis on **reusability**, **configuration over customization**, and templated implementations.

Deployment Architecture and Automation

- Develop and maintain automated deployment strategies (e.g., using CI/CD pipelines, containers, cloud images).
- Ensure consistency of environments across units by applying **DevOps and IaC principles**.

Reference Architecture Development

- Define **blueprint architectures** for infrastructure, security, networking, and applications.
- Maintain architecture artifacts to facilitate rapid rollout and onboarding of new units.

Governance and Compliance Oversight

- Establish and enforce **IT governance policies** to maintain standardization.
- Monitor compliance with **corporate security**, **operational**, **and data handling standards**.

Support Scalability and Localization (Where Needed)

- Provide mechanisms for **minor localization** (e.g., local taxes, languages) without deviating from the core architecture.
- Architects must balance **global consistency** with **necessary regional adaptability**.

Aspect	Characteristic
IT Architecture Style	Centralized standards with local instances
Technology Choices	Uniform across all business units
Integration Level	Low – limited to central reporting or oversight
Data Sharing	Minimal across units; centralized reporting only
IT Architect Focus	Standardization, repeatability, deployment automation
EA Role	Strong governance, template enforcement, compliance monitoring
Key Challenges	Managing local exceptions, maintaining global uniformity, automating scale

7.1.4: Unification Model

The **Unification Model** is characterized by both **high integration** and **high standardization**, where business units are **tightly connected** and

operate with **consistent processes** (Figure 5). This approach emphasizes **efficiency** and a **unified operating framework**, making it particularly suitable for **centralized organizations** like **retail chains** and **airlines**. For instance, retail chains benefit from standardized **point-of-sale** and **inventory systems** across all locations, while airlines rely on a **consistent customer experience** and **streamlined operations**.

image by ollo from istock

Pros:

- **Efficiency**: Streamlined operations and standardized systems lead to significant cost efficiencies.
- **Consistency**: A unified operating approach ensures consistent quality and improves customer satisfaction.
- **Centralization**: Facilitates easier management and oversight across business units.

Cons:

- **Limited Flexibility**: This model may restrict the adaptability of individual units to respond to specific market demands.

- **High Investment**: Implementing unified systems and processes requires substantial initial investment.
- **Potential for Resistance**: Employees may resist standardization efforts if they feel it limits their autonomy or creativity.

In summary, while the **Unification Model** can enhance efficiency and uniformity across an organization, it may also present challenges in flexibility and require significant investments to implement successfully.

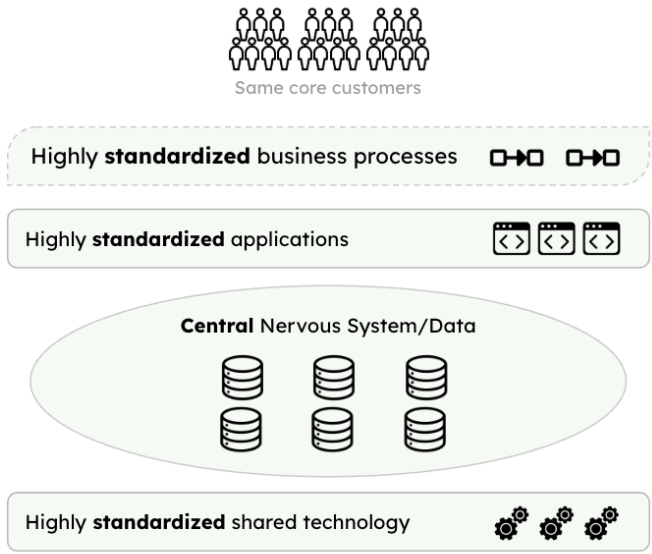

***Figure 5**: A high-level core diagram of the unification operating model.*

Here is a structured overview of the typical **IT architecture** and **IT architect responsibilities** for the **Unification Operating Model**, which is defined by **high integration** and **high standardization**—optimal for centralized, process-driven organizations like **retail chains** and **airlines**:

Centralized, Monolithic or Modular Core Systems

- Enterprise-wide platforms (e.g., ERP, POS, CRM, HRM) are **standardized and shared across all business units**.

- Systems are **tightly integrated**, often forming a **single source of truth**.

High Process Standardization

- Uniform processes for finance, HR, inventory, procurement, and customer service are enforced enterprise-wide.
- **Business process management (BPM)** systems or workflows may be used to ensure compliance.

Strong Data Integration and Centralization

- Centralized **data warehouses** or **data lakes** feed enterprise analytics and decision-making.
- Real-time data synchronization across departments and locations (e.g., inventory availability across all stores).

Unified Security and Governance

- Standardized security policies, identity and access management, and compliance frameworks are applied universally.
- **Central governance bodies** oversee architecture, data, and infrastructure decisions.

Cloud-First or Hybrid Cloud Architecture

- Common use of **enterprise-grade cloud platforms** for scalability, availability, and centralized control.
- Cloud-based or hybrid architecture enables consistent deployment across global locations.

Enterprise Architecture (EA) Leadership

- Architects define and enforce **standard architectures**, platforms, and design patterns.
- Ensure systems are **modular and scalable**, but not diverging from enterprise-wide standards.

System Integration Architecture

- Architect and maintain **enterprise integration frameworks** (e.g., ESB, API management platforms, ETL pipelines).
- Ensure **seamless data flow** between systems (e.g., POS to ERP, CRM to marketing platforms).

Governance and Standardization

- Drive **compliance with architectural standards** and system usage policies.
- Define **architecture roadmaps**, governance models, and technology adoption plans.

Infrastructure and Deployment Strategy

- Design centralized, secure, and resilient infrastructure supporting **multi-site operations**.
- Often involves **centralized DevOps**, CI/CD pipelines, and infrastructure automation.

Optimization and Efficiency Engineering

- Continual improvement of IT systems to increase **operational efficiency**, **reduce technical debt**, and **optimize resource use**.
- Collaborate with operations teams to ensure that IT aligns tightly with business goals.

Aspect	Characteristic
IT Architecture Style	Centralized, standardized, and highly integrated
Technology Choices	Uniform enterprise-wide platforms
Integration Level	Very high – all systems and data interconnected
Data Sharing	Seamless, real-time across all units

Aspect	Characteristic
IT Architect Focus	System unification, standardization, integration, governance
EA Role	Strong leadership role in architectural decisions and enforcement
Key Challenges	Scalability, change management, high upfront investment, limited local flexibility

7.1.5: Choosing an Operating Model

Each operating model offers a distinct balance of standardization and integration. The choice depends on the organization's strategy, structure, and goals. Implementing the right operating model enables the organization to achieve operational efficiency and strategic alignment.

- **Customer Needs**: Do customers expect consistency or customization?
- **Products/Services**: Are products/services similar across units or vary widely?
- **Data Requirements**: Is real-time data sharing critical for business success?
- **Cost vs. Agility**: What's the trade-off between standardization and flexibility?

7.2: Global vs. Local Business Flexibility

Global flexibility allows organizations to adapt to external changes affecting their overall strategy, while local flexibility enables specific business units to respond quickly to their unique needs.

7.2.1: Global Flexibility

Global flexibility refers to the organization's ability to adapt and respond to external changes, opportunities, and challenges that affect the **entire enterprise** or its overarching strategy. An example is a **global retailer** implements a **centralized inventory management system** that provides real-time stock visibility across all regions. This system enables the company to **adapt to global supply chain disruptions** or **launch new global initiatives** efficiently.

image by metamorworks from istock

Global flexibility is characterized by an **enterprise-wide perspective** that emphasizes solutions, processes, and systems designed to enable adaptability across the organization. This approach ensures that **shared**

capabilities, such as common data standards and unified platforms, are structured to support rapid pivots or innovation.

Key enablers of global flexibility include **standardized platforms** and technology stacks, **unified governance** and decision-making frameworks, and **enterprise-wide data models** with robust integration. These elements create a cohesive foundation that simplifies large-scale transformations and global initiatives. By standardizing processes and technologies, organizations can **reduce redundancy and complexity** while **improving scalability** across different regions or business units.

However, achieving global flexibility is not without its challenges. It can sometimes **feel rigid** to individual business units or regions that require specialized solutions. Additionally, centralized control may **slow down localized innovation**, potentially hindering responsiveness to unique regional needs.

7.2.2: Local Flexibility

Local flexibility is the ability of **individual business units**, departments, or regions to respond quickly to their **specific needs, opportunities, and challenges**. An example is a **multinational company** operating in different countries that allows its **regional offices** to **customize marketing campaigns** based on local cultural preferences and consumer behavior.

image by _marqs from istock

Local flexibility is defined by a unit-specific perspective that prioritizes **empowering local teams** with **tailored processes and solutions** to address unique demands. This approach emphasizes decentralization and grants autonomy to individual business units or regions, fostering a focus on localized needs and innovation.

The foundation of local flexibility lies in modular systems that **support customization**, **decentralized governance and decision-making frameworks**, and **well-defined APIs** or interfaces that enable seamless integration between local and global systems. These elements allow organizations to adapt quickly to specific market demands or functional requirements.

This **flexibility enhances customer satisfaction** by addressing local or niche needs, encourages **innovation through experimentation**, and enables **agility in smaller-scale contexts**. However, it also presents challenges, such as the risk of **fragmentation or duplication of efforts** across the organization. Additionally, localized approaches can result in **inefficiencies or integration** difficulties when attempting to scale solutions at an enterprise-wide level.

7.2.3: Balancing Global and Local Flexibility

The real challenge lies in balancing global and local flexibility to maximize the organization's effectiveness.

This balance depends on the operating model and the nature of the business.

- **When to Emphasize Global Flexibility**: For processes or capabilities that drive enterprise-wide efficiency, compliance, and scalability (e.g., finance, supply chain, IT infrastructure). In industries where consistency and standardization are critical (e.g., pharmaceuticals, aviation).
- **When to Emphasize Local Flexibility**: For customer-facing processes or where regional differentiation is a competitive advantage (e.g., marketing, sales, customer service). In highly decentralized industries or organizations with diverse product portfolios.
- **Enabling Both**: implement a **modular architecture** where core systems are standardized, but local modules can be customized. Use a **federated governance model** that balances central oversight with regional autonomy. Build **shared platforms** with configurable features to address both global and local needs.

In summary, global flexibility provides consistency and scalability, allowing organizations to respond effectively to strategic changes across the enterprise. In contrast, local flexibility drives innovation and responsiveness by addressing regional or unit-specific requirements. Organizations that balance these two forms of flexibility develop resilient and adaptable architectures, enabling them to support strategic goals at every level.

7.3: Stages of Business Strategy and IT Alignment

The Enterprise Architecture as Strategy book identifies four stages of Business Strategy and IT Alignment, with increasing levels of strategic value:

- **Business Silos**: Independent systems and processes with minimal integration.
- **Standardized Technology**: Common technology platforms but still fragmented business processes.
- **Optimized Core**: Standardized and integrated processes aligned with the operating model.
- **Business Modularity**: Modular, reusable components enabling strategic agility and innovation.

In this context, the concepts of **global flexibility** and **local flexibility** refer to different ways organizations balance standardization and customization to achieve efficiency and adaptability (Figure 6).

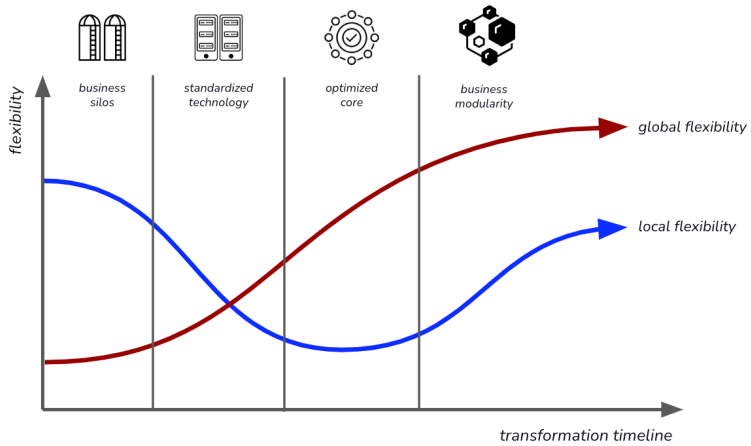

***Figure 6**: Balancing local and global business flexibility in different stages of Business Strategy and IT Alignment.*

Each stage reflects the level of standardization, integration, and strategic value IT systems provide. Progressing through these stages enables an organization to move from fragmented IT systems to a flexible, reusable architecture supporting strategic goals.

7.3.1: Stage 1: Business Silos

In the **Business Silos** stage, IT systems and business processes are developed independently within business units, resulting in fragmentation and inefficiencies. There is little to no integration or standardization, and each business unit prioritizes its goals and initiatives.

image by mathisworks from istock

In this approach, systems are designed specifically to meet the needs of individual business units, with minimal consideration for enterprise-wide goals. Data is often stored in silos, limiting the ability to share information across the organization. IT functions primarily as a support role, addressing local requirements rather than serving as a strategic enabler.

This localized focus presents several challenges, including significant duplication of processes and systems, a lack of visibility and consistency across the organization, and difficulties in scaling or responding effec-

tively to enterprise-wide initiatives. The primary emphasis remains on addressing the immediate needs of individual business units.

7.3.2: Stage 2: Standardized Technology

At the **Standardized Technology** stage, organizations begin consolidating and standardizing IT infrastructure across the enterprise. This standardization reduces complexity and costs but still lacks a focus on business process integration.

image by mathisworks from istock

This approach is characterized by a centralized IT infrastructure that relies on shared platforms, tools, and technologies. While business units maintain operational independence, they utilize standardized technology, resulting in reduced IT costs through economies of scale and streamlined support.

Despite these benefits, challenges persist. Business processes often remain fragmented, as standardization efforts primarily focus on technology rather than aligning processes. Additionally, strategic alignment between IT and business objectives is limited. This model's primary focus is on achieving cost efficiency and ensuring IT reliability through shared infrastructure.

7.3.3: Stage 3: Optimized Core

The **Optimized Core** stage marks a shift from a technology-driven focus to a business process-driven approach. Organizations standardize and integrate critical processes aligned with the operating model to improve efficiency and coordination across business units.

image by mathisworks from istock

This model emphasizes standardized and integrated core business processes, such as finance, supply chain, and HR, to ensure consistency across the enterprise. With enterprise-wide data and process alignment, organizations can make more informed decisions, and IT takes on a strategic role in optimizing operations to support the overall operating model.

The benefits include enhanced operational efficiency, scalability, and an improved ability to respond effectively to enterprise-wide initiatives. However, the approach also faces challenges, such as resistance from business units to standardize and integrate processes and the need for significant investment in systems and organizational change. The primary focus is aligning business processes with enterprise-wide goals to drive operational efficiency.

7.3.4: Stage 4: Business Modularity

The **Business Modularity** stage represents the highest business and IT alignment level. Organizations adopt a modular approach, creating reusable business and technology components that can be quickly reconfigured to support innovation and changing strategies.

image by mathisworks from istock

This approach is defined by modular and reusable business and IT components, such as APIs, microservices, and plug-and-play processes. It provides high flexibility and agility, enabling the organization to adapt swiftly to new opportunities or disruptions. Business units are empowered to innovate while leveraging shared enterprise resources.

The benefits include increased strategic agility, rapid innovation and responsiveness to market changes, and scalability with lower incremental costs. It also fosters strong alignment between business and IT. However, the approach presents challenges, such as the complexity of designing and maintaining modular components and the need for a mature governance model to manage shared resources effectively. The primary focus is achieving strategic agility and enabling rapid innovation through modularity.

7.3.5: Progressing Through the Stages

Advancing through these stages requires deliberate effort, including investments in technology, process reengineering, and governance. Organizations typically move sequentially but may need to revisit earlier stages when addressing gaps or implementing new strategies.

- **From Business Silos to Standardized Technology**: Focus on reducing IT complexity and costs by consolidating infrastructure.
- **From Standardized Technology to Optimized Core**: Shift to aligning processes and systems with the operating model to enhance efficiency.
- **From Optimized Core to Business Modularity**: Adopt modular architecture to achieve strategic agility and enable rapid innovation.

The maturity stages illustrate the progression of enterprise architecture (EA) from a fragmented IT environment to a strategic enabler of agility and innovation. Each stage builds upon the last, progressively strengthening the alignment between IT and business while enhancing the organization's capacity to execute its strategic objectives. Advancing to higher maturity stages demands a blend of technological advancements, organizational adjustments, and cultural shifts. Strong leadership and governance are essential to navigate resistance and manage the growing complexity associated with this evolution.

By progressing through these stages, organizations can achieve operational excellence and strategic agility, enabling them to thrive in a rapidly changing business environment.

7.4: To Probe Further

- Enterprise Architecture as Strategy: Creating a Foundation for Business Execution[2] by Jeanne W. Ross, Peter Weill, and David C. Robertson, 2006.
- A New Way to Think: Your Guide to Superior Management Effectiveness[3] by Roger L. Martin, 2022.

[2] https://store.hbr.org/product/enterprise-architecture-as-strategy-creating-a-foundation-for-business-execution/8398

[3] https://www.amazon.com/New-Way-Think-Management-Effectiveness-ebook/dp/B09KHKWB7B

7.5: Questions to Consider

- *Which business operating model (Diversification, Coordination, Replication, or Unification) best describes your organization's current approach? Is it the right fit for your strategy?*
- *How does your organization balance global and local flexibility in its IT and business operations? Where are the biggest challenges?*
- *What stage of IT and strategy alignment is your company currently at—Business Silos, Standardized Technology, Optimized Core, or Business Modularity?*
- *How well is your IT architecture aligned with your company's business strategy, and what gaps exist?*
- *What are the trade-offs your organization faces between efficiency (global standardization) and responsiveness (local flexibility)?*
- *How do governance, security, and compliance factor into your company's IT-business alignment efforts?*
- *What challenges arise when transitioning from fragmented IT systems to a more integrated enterprise architecture?*
- *How does your company ensure that its technology investments support both immediate business needs and long-term strategic goals?*
- *What role does modular architecture play in improving your organization's agility and ability to innovate?*
- *How can IT leaders and business executives better collaborate to ensure continuous realignment between technology and strategy?*

8: Outsourcing Strategies

image by nes from istock

IN THIS SECTION, YOU WILL: Learn about outsourcing strategies—strategic outsourcing, co-sourcing, and transactional outsourcing—help.

KEY POINTS:

- Strategic Outsourcing: Long-term partnerships that drive innovation and align with business goals, requiring strong governance and integration with IT architecture.
- Co-Sourcing: A collaborative model where internal and external expertise is blended for shared responsibility, requiring well-defined roles, governance structures, and architectural oversight.
- Transactional Outsourcing: A cost-driven approach focused on routine, non-strategic tasks, emphasizing efficiency, automation, and seamless integration with enterprise systems.
- IT Architecture's Role: Ensures strategic alignment, interoperability, governance, risk management, and smooth integration of outsourced capabilities.
- Strategy Selection: Organizations should choose an outsourcing model based on their operational maturity, business objectives, and the criticality of the outsourced function.

Outsourcing Strategies examine various approaches to utilizing **external providers**, guided by **business goals**, **operational needs**, and the desired level of **control**. In *Enterprise Architecture as Strategy*, the authors identify three key outsourcing strategies: **strategic outsourcing**, **transactional outsourcing**, and **co-sourcing**. Each strategy represents a different method for organizations to **leverage external providers**, depending on their **business objectives**, **operational requirements**, and **desired control levels**.

- **Strategic outsourcing** aligns closely with an organization's **long-term goals** and drives **innovation** by integrating with the overall **business strategy**.
- In contrast, **transactional outsourcing** emphasizes **cost reduction** and **operational efficiency** for **routine**, **non-strategic tasks**.
- **Co-sourcing** offers a **balanced approach**, encouraging **collaboration** and **shared responsibility** for **critical or complex processes**, while combining the strengths of both **internal teams** and **external providers**.

Each strategy serves **distinct purposes** and should be selected based on the organization's **needs**, **priorities**, and the specific **function being outsourced**. These strategies also reflect the organization's **maturity in enterprise architecture** and its **capability to manage relationships** with external providers.

An **IT architecture practice** plays a crucial role in shaping outsourcing strategies by ensuring **alignment with enterprise goals**, maintaining **governance**, and facilitating **seamless integration**. Whether through long-term strategic outsourcing, collaborative co-sourcing, or cost-driven transactional outsourcing, IT architects provide the necessary **frameworks**, **standards**, and **oversight** to **maximize the value** of external partnerships.

8.1: Strategic Outsourcing

Strategic outsourcing involves forming **long-term partnerships** with external providers to achieve **key business objectives**. This strategy is closely aligned with the organization's **operating model** and **strategic goals**, with providers contributing to **operational efficiency**, **innovation**, and **value creation**.

image by metamorworks from istock

The focus of this approach is on **high-value**, **strategic functions** that are **critical to the organization's success**. It emphasizes **collaboration** and **alignment** between the organization and its outsourcing partner, often involving a **shared approach to risks and rewards**.

Examples of strategic outsourcing include partnering with **technology providers** to **co-develop new software platforms** or digital products, and outsourcing **advanced analytics** or **research and development (R&D)** to specialized firms that enhance internal capabilities.

The advantages of this model include access to **specialized expertise** and **innovation**, increased **strategic agility** through external resources, and a stronger focus on **core competencies**. However, it also presents

challenges, such as the need for significant **investment in relationship management** and **governance**, as well as potential **risks related to dependency** on external providers if not carefully managed.

In addition, **IT architecture** plays a crucial role in ensuring these partnerships align with the organization's **technology and business strategy**. The role of IT architecture in strategic outsourcing typically involves:

- **Strategic Alignment**: Defining **architectural principles** and **technology roadmaps** to guide outsourced solutions in line with **business goals**.
- **Capability Integration**: Ensuring that outsourced **technology capabilities** integrate seamlessly with **internal systems** and **enterprise platforms**.
- **Innovation Enablement**: Facilitating **co-development efforts** and ensuring architectures are **flexible** and support **emerging technologies**.
- **Governance and Risk Management**: Establishing **frameworks** for **architectural governance**, **security**, **compliance**, and **resilience**—especially when external entities manage **critical workloads**.
- **Technology Standards and Interoperability**: Defining **API strategies**, **data exchange protocols**, and **cloud adoption frameworks** to ensure **interoperability** between internal and outsourced systems.

8.2: Co-Sourcing

Co-sourcing is a **hybrid approach** in which the organization and the outsourcing provider **share responsibility** for a particular function or process. This model combines the advantages of the provider's **expertise** with the organization's **internal knowledge**, ensuring that both parties work together to **achieve common goals**.

image by scyther5 from istock

Co-sourcing emphasizes **close collaboration** between internal teams and the outsourcing provider, making it particularly suitable for **complex or critical processes** where retaining some degree of **control** is essential. **Governance structures** are established to clearly define **roles**, **responsibilities**, and **decision-making processes**.

Examples of co-sourcing include **co-developing IT systems**, where the organization retains control over **architectural decisions** while the provider manages **implementation**. Another example is **managing cybersecurity**, where both parties share responsibilities: the provider offers **specialized tools and monitoring**, while internal teams focus on **escalation** and **strategic decisions**.

This approach effectively blends **external expertise** with **internal knowledge**, leading to **enhanced performance** and **alignment with organizational goals**. It provides **flexibility** to adapt to changing requirements while maintaining **control over key aspects**. This **collaborative relationship** fosters **strong partnerships** and **mutual trust**.

However, the model also presents challenges, such as the need for **clear governance** and **effective communication** to avoid **conflicts** or **inefficiencies**. It can be **more complex to manage** compared to purely outsourced or in-house solutions. The success of this model hinges on the provider's ability to **integrate seamlessly** with the organization's **culture and processes**.

Co-sourcing effectively combines **internal and external expertise**, **sharing responsibility** over **key processes** or **technology functions**. The role of **IT architecture** in co-sourcing typically involves:

- **Architectural Control & Oversight**: Defining and enforcing architecture standards while allowing flexibility for co-sourced partners to contribute effectively.
- **Collaboration & Integration Models**: Developing blueprints for collaboration that clarify the boundaries between in-house and outsourced responsibilities (e.g., specifying where internal teams manage design and where providers handle implementation).
- **Security & Compliance Frameworks**: Establishing shared security and compliance standards to ensure that co-sourced solutions align with regulatory requirements.
- **Scalability & Adaptability**: Designing architectures that support evolving business needs, ensuring co-sourced capabilities can scale or be transitioned in-house as necessary.
- **Governance & Decision Rights**: Creating governance structures that balance decision-making authority between internal architects and external providers.

8.3: Transactional Outsourcing

Transactional outsourcing is a **tactical approach** aimed at outsourcing **routine**, **non-strategic tasks** or processes to external providers. The primary objective is to achieve **cost efficiency** by transferring **repetitive** or **easily standardized functions** to these providers.

image by nanostockk from istock

This outsourcing model is typically **short-term** or **contract-based**, with clearly defined **deliverables** and **performance metrics**. It emphasizes **cost reduction** and **operational efficiency**, focusing mainly on **non-core activities** while maintaining **minimal collaboration** or **strategic alignment** between the organization and the provider.

Common examples of transactional outsourcing include functions like **payroll processing**, **data entry**, **IT helpdesk services**, and contracting external firms for **temporary staffing** or **basic maintenance tasks**.

The advantages of this approach include **immediate cost savings**, improved **operational efficiency**, and the ability to **free up internal resources** for strategic priorities. It is relatively **straightforward to implement** due to clear **contractual obligations** that outline expectations.

However, this model also presents challenges. It often provides **limited long-term value** beyond cost savings. **Quality may decline** if the

provider prioritizes cost reduction over service excellence, and there is **little flexibility** to adapt outsourced processes to **evolving organizational needs**.

In **transactional outsourcing**, the focus is on **cost reduction** through the outsourcing of **routine, non-strategic tasks**. **IT architecture** plays a critical role in ensuring these external services **integrate smoothly** without introducing **risk or complexity**. The key functions of IT architecture in this context include:

- **Process Standardization**: Establish **structured workflows** and **automation strategies** to optimize repetitive outsourced tasks.
- **Service Integration & API Management**: Implement **service-oriented architectures** and **API gateways** to facilitate **secure** and **efficient integration** of outsourced services.
- **Vendor Evaluation & Compliance**: Help **select outsourcing providers** that meet **architectural** and **security standards**.
- **Performance Monitoring & Optimization**: Define **metrics** and **monitoring frameworks** to ensure outsourced services meet **service-level agreements (SLAs)**, such as uptime, response time, and data accuracy.
- **Risk Mitigation & Exit Strategy**: Create **architectures** that enable **smooth transitions** if the organization needs to **switch vendors** or **bring outsourced functions back in-house**.

8.4: To Probe Further

- Enterprise Architecture as Strategy: Creating a Foundation for Business Execution[1] by Jeanne W. Ross, Peter Weill, and David C. Robertson, 2006.

[1] https://store.hbr.org/product/enterprise-architecture-as-strategy-creating-a-foundation-for-business-execution/8398

8.5: Questions to Consider

- *Which outsourcing strategy—strategic outsourcing, co-sourcing, or transactional outsourcing—best aligns with your organization's current business goals and operational needs? Why?*
- *How does your organization currently balance control, collaboration, and cost efficiency in outsourcing decisions? Could this balance be improved?*
- *What challenges has your organization faced when working with external providers, and how could a stronger IT architecture practice help mitigate these challenges?*
- *In what ways can IT architecture enhance the governance and integration of outsourced services within your enterprise technology landscape?*
- *How does your organization ensure that outsourced capabilities remain aligned with its long-term business and technology strategy?*
- *What key factors should be considered when selecting an outsourcing provider to ensure compatibility with your organization's IT and business frameworks?*
- *How does your organization handle risk management, security, and compliance when engaging with external service providers?*
- *What role should internal IT teams play in managing and overseeing outsourced functions to maximize value and minimize risks?*
- *If your organization were to transition to a different outsourcing model (e.g., from transactional outsourcing to co-sourcing), what challenges and benefits would you anticipate?*
- *How can your organization build stronger, more resilient partnerships with external providers to drive innovation while maintaining flexibility and strategic agility?*

Part III: On Human Complexity

9: Expanding the Architect's Toolkit: Learning From Other Fields

image by worawee meepian from istock

IN THIS SECTION, YOU WILL: Get a summary of several resources that I use as inspiration for running the Grounded Architecture practice in complex organizations.

A holistic approach is essential for achieving success in the intricate and fast-paced world of IT architecture, especially within complex organizations. Traditional IT architecture literature often fails to address the multifaceted challenges architects face today. To bridge this gap, I have curated a selection of resources that provide a broader perspective, drawing inspiration from social sciences, behavioral sciences, product management, and political sciences. These resources offer valuable insights and practical strategies for running the Grounded Architecture practice in complex organizational environments.

- **Economic Modeling With ROI and Financial Options**: I sketch two methods of determining the economic value of technology investments and architecture: the return on investment metaphor and the financial options metaphor.
- **Architecture in Product-Led Organizations: Learning From Customer-Centric Fields**: Effective product development is a cornerstone of organizational success. 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 "Product Operations" by Melissa Perri and Denise Tilles. The build trap occurs when businesses focus too much on their product's features and functionalities, overlooking customers' needs and preferences. Product Operations is the discipline of helping product management scales well, surrounding teams with essential inputs to set strategy, prioritize, and streamline ways of working.
- **Decision Intelligence in IT Architecture: Learning From Data, Social, and Managerial Fields**: Decision intelligence is a burgeoning field that combines data science, social science, and managerial science to transform information into actionable insights. In the context of IT architecture, decision intelligence is becoming increasingly vital. It equips architects with the tools and frameworks to analyze data effectively, foresee potential outcomes, and make strategic decisions that align with organizational goals. Understanding decision intelligence is essential for architects who aspire to drive innovation and efficiency in their projects.
- **How Big Transformations Get Done: Learning From Mega-Projects**[1]: IT transformation projects face similar challenges as

[1] big-projects

other mega-projects, often failing due to cognitive biases and poor planning. However, applying key lessons from successful projects—such as risk mitigation, modular design, and stakeholder engagement—can significantly improve their outcomes.

By exploring these diverse resources, IT architects can significantly enhance their ability to drive strategic initiatives within their organizations. These insights provide a robust framework for developing a grounded and effective an architecture practice, enabling architects to address the unique challenges of their roles with confidence and precision. Whether you are an aspiring architect or a seasoned professional, these perspectives will offer valuable guidance on navigating and excelling in the dynamic field of IT architecture.

10: Economic Modeling With ROI and Financial Options: Learning From the Finance Field

image by microstockhub from istock

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

KEY POINTS:

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

Decision-making in the corporate world is frequently an **economic risk exercise**. Financial and risk modeling is like the crystal ball of the corporate world. It helps organizations make intelligent decisions, like choosing between investing in a new project or finally fixing the office coffee machine. These models forecast financial performance and assess economic scenarios, ensuring companies aren't just throwing darts in the dark.

Economic and risk modeling is a game-changer in resource allocation. By predicting future trends and disruptions, these models help organizations **use their resources wisely**, for instance, to ensure they don't run out of coffee (a critical issue for any IT business). It's all about efficient resource management and avoiding those 'Oops, we should have seen that coming' moments.

These models majorly upgrade strategic planning, helping companies **anticipate challenges and opportunities** instead of reacting like a cat to a cucumber. Identifying risks before they bite means organizations can implement preventative measures, keeping things running smoothly.

Economic and risk modeling is the secret weapon for staying ahead of the game, achieving long-term goals, and ensuring the office party budget doesn't get blown on a single extravagant cake.

Organizations conduct financial and risk modeling exercises, such as ROI calculations, for several key reasons:

- **Decision-Making Support**: Evaluate investments and compare alternatives to allocate resources effectively.

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

As financial and risk modeling is essential in any organization, architects frequently need to answer questions about the **(economic) value of technology investments and architecture**. Answering this question is a crucial skill for any senior architect. Still, it may take much work to answer this seemingly harmless question concisely and convincingly to a non-technical audience without sounding like a techie version of Shakespeare.

Good architecture requires some investment. This investment is time and effort spent implementing an architecture pattern, reducing technical debt, or refactoring code to align with our architecture. Consequently, we need to explain the expected value of this investment. It's all about showing that a little investment now will save a lot of headaches—and money—later.

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

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

10.1: 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**.

image by nicoelnino from istock

A high ROI means the investment's gains compare favorably to its cost. As a performance measure, you can use ROI to evaluate an investment's efficiency or compare the efficiencies of several different investments (Figure 1).

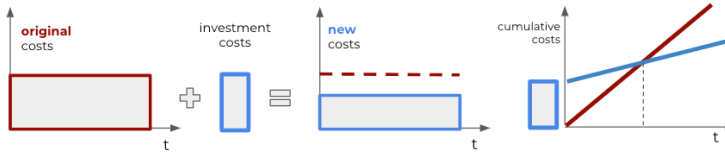

Figure 1: *An illustration of the ROI metaphor. Investment leads to lower costs or higher value. It takes some time to reach a break-even point when an additional value has compensated for the investment. After the break-even point, we earn more than without the investment.*

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

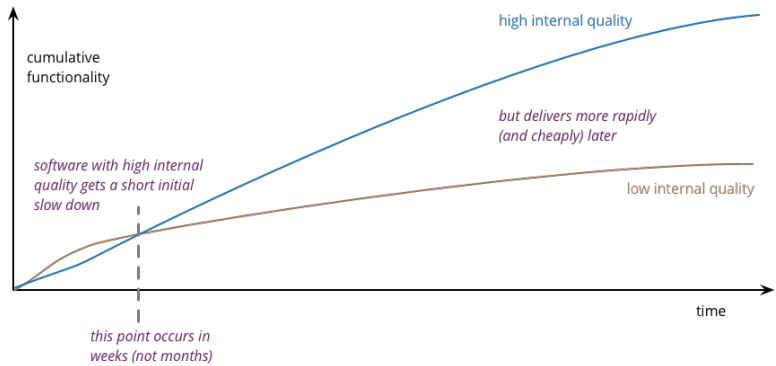

Figure 2: *Software with high internal quality gets a short initial slowdown but delivers more rapidly and cheaply later (source martinfowler.com/articles/is-quality-worth-cost.html).*

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

[1] https://martinfowler.com/articles/is-quality-worth-cost.html

not make such investments. The following section explains why.

10.2: 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.

image by olivier le moal from istock

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).

[2] https://en.wikipedia.org/wiki/Black%E2%80%93Scholes_model

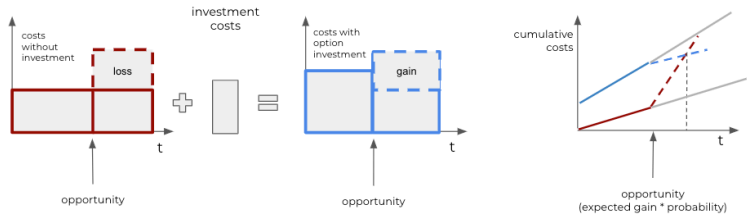

Figure 3: *An illustration of the financial option metaphor. Options have a price, leading to higher initial costs. However, if an opportunity can generate more value, we gain additional profit (or lose it if we do not invest).*

Applying the financial option metaphor to IT architecture, we can argue that **buying options gives the business and IT a way to defer decisions**. Gregor Hohpe gives an example of the server size you need to purchase for a system. If your application is architected to be horizontally scalable, you can defer this decision: additional (virtual) servers can be ordered later at a known unit cost.

Another example of an IT option is architecting your system to **separate concerns**. For instance, deciding early what authentication mechanism an application should use may be challenging. A system that properly 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.

10.3: Communication Frameworks

UNESCO's manual for investigative journalism[3] says that *"The facts do not tell the story. The story tells the facts."* The same applies to any financial analysis. Good data and analysis can have zero impact if they are not communicated in a way that people understand and lead to action.

10.3.1: General Framework For Communicting Value of IT Investments

Here, I share communication frameworks I developed and used to explain holistically the economic value of architecture and technology investment (Figure 4).

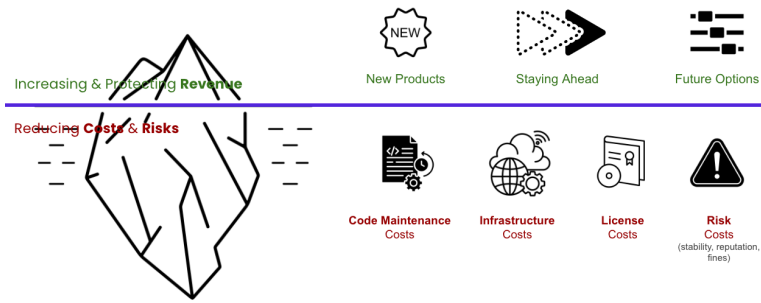

Figure 4: *A framework for discussing investments and options.*

I separate the value of investments in two buckets:

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

[3] https://unesdoc.unesco.org/ark:/48223/pf0000193078

10.3.1.1: Top Line: Increasing and Protecting Revenue

Increasing and protecting revenue investments have three forms.

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

Investments needed to stay ahead. This type of investment is a less obvious way to protect and increase revenue. It boils down to the fact that you cannot stop developing your product as the rest of the world moves on. As the saying goes, "**It takes all the running you can do to keep in the same place**." For instance, you must keep essential features in parity with the competition, your system must comply with changes in regulations, and your UX must be modern.

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

10.3.1.2: Bottom Line: Reducing Costs and Risks

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

Investments to reduce maintenance costs need to ensure that your code is easy to understand, change, and test. Such investments directly reduce your most significant cost, people costs, as code that is easy to maintain requires fewer people. Alternatively, you can look at these investments as a way to spend more effort on innovation and creating new revenue streams rather than merely keeping the systems in the air. Figure 5 illustrates what may happen if you do not invest. As systems grow in size and complexity, more developers are needed to maintain them. If the system is not easy to maintain, people will avoid touching code as they can easily break it. This situation will lead to a workaround

(such as copying and pasting code and diverse hacks). These inefficient workarounds further increase the size and complexity of code, requiring even more developers to maintain it. And the vicious, expensive cycle continues.

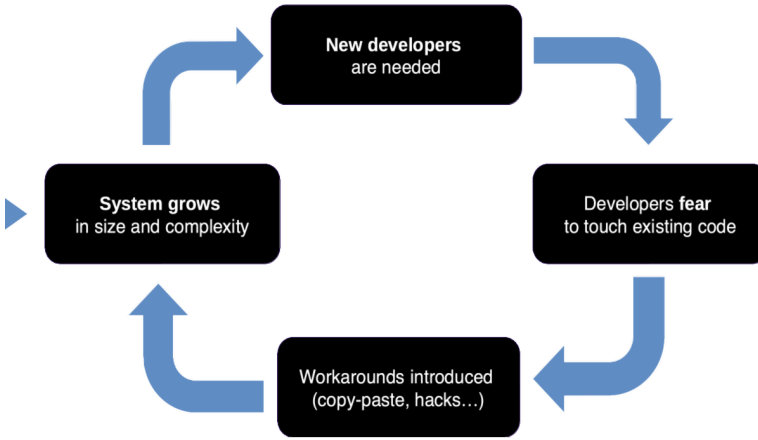

Figure 5: *A downward spiral of poorly maintainable code.*

Investments in reducing infrastructure costs reduce spending and, if successful, are more directly visible. Such investments could take the form of redesigning your application to be more elastic, scaling up and down with minimal overhead. They could also create more transparency to have a precise image of all cost drivers and mechanisms to react quickly to any undesirable cost increases.

Investments in reducing license and vendor costs ensure that there is no unnecessary diversity of technologies and vendor contracts and that you can leverage economies of scale, as having fewer vendors with more users enables negotiating more favorable contracts.

Investments in reducing risk costs. When your system is down, your business is disrupted, and you lose revenue. According to diverse studies[4], 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

[4] https://www.atlassian.com/incident-management/kpis/cost-of-downtime

importance of continuing such investments. Or, as noted by Repenning and Sterman "Nobody Ever Gets Credit for Fixing Problems that Never Happened[5]".

10.3.2: The Tech Debt Reduction ROI Framework

In addition to the general framework discussion in this section, I frequently used the following framework to visually communicate how increasing technical debt progressively reduces profit through added operating, delay, and instability costs, emphasizing the importance of managing technical debt to preserve profitability.

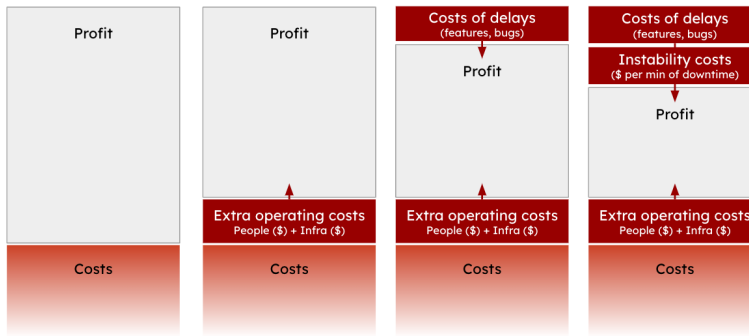

Figure 6: *A framework for discussing ROI of tech debt reduction from a profitability standpoint.*

Figure 6 illustrates a framework for understanding the ROI (Return on Investment) of reducing technical debt, showing the impact on profit and costs across three progressive dimensions:

1. **Extra Operating Costs**: Technical debt can increase operating costs due to inefficiencies (e.g., complexity or lack of automation may require more people and time to make changes), reducing profit by raising expenses.
2. **Costs of Delays**: This dimension represents the loss of profit related to delays. Technical debt can impact the ability to deliver new features or fix issues promptly.

[5] https://web.mit.edu/nelsonr/www/CMR_Getting_Quality_v1.0.html

3. **Instability Costs**: Tech debt is a risk that can lead to less stable systems, with more frequent and longer downtimes and errors. Instability and downtime further reduce profit.

This framework shows the compounded effect of technical debt, where extra operating costs increase overall costs, and instability and downtime further reduce profit.

10.4: To Probe Further

- Architecture: Selling Options[6], by Gregor Hohpe, 2016
- Is High Quality Software Worth the Cost?[7], by Martin Fowler, 2019
- Don't get locked up into avoiding lock-in[8], by Gregor Hohpe, 2019

[6] https://architectelevator.com/architecture/architecture-options/
[7] https://martinfowler.com/articles/is-quality-worth-cost.html
[8] https://martinfowler.com/articles/oss-lockin.html

10.5: Questions to Consider

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

11: Architecture in Product-Led Organizations: Learning From Customer-Centric Fields

image by tevarak from istock

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

KEY POINTS:

- When it comes to product development, I generally recommend two resources for architects: "Escaping the Build Trap: How Effective Product Management Creates Real Value" by Melissa Perri and "Product Operations" by Melissa Perri and Denise Tilles.
- The build trap occurs when businesses focus too much on their product's features and functionalities, overlooking customers' needs and preferences.
- Product Operations is the discipline of helping product management scales well, surrounding teams with essential inputs to set strategy, prioritize, and streamline ways of working.

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

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

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

- "Escaping the Build Trap: How Effective Product Management Creates Real Value[1]" by Melissa Perri, and
- "Product Operations[2]" by Melissa Perri and Denise Tilles.

[1] https://www.goodreads.com/book/show/42611483-escaping-the-build-trap
[2] https://www.productoperations.com/

Both of these sources provide several things for architects:

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

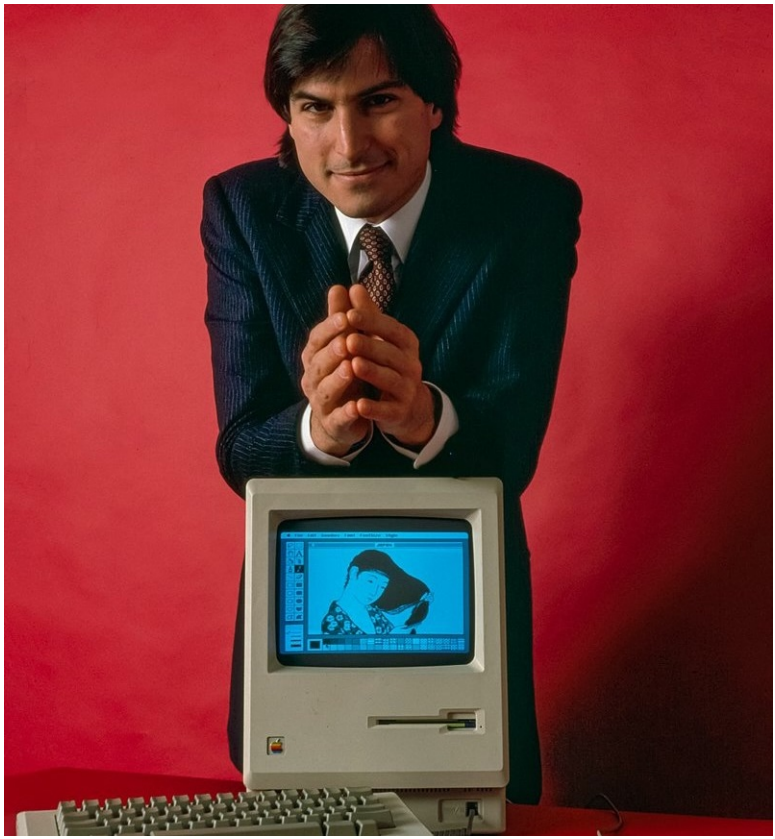

image by wikimedia commons

11.1: Product Development

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

11.1.1: 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.*

11.1.2: 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.

11.1.3: 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.

11.1.3.1: The Mini-CEO

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

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

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

image by khosrork from istock

11.1.3.2: The Former Project Manager

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

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

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

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

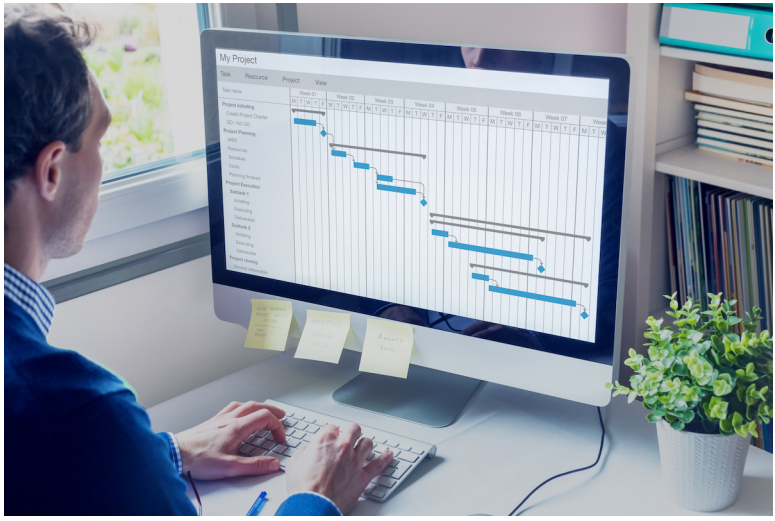

image by nicoelnino from istock

11.1.3.3: 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 gorodenkoff from istock

[3] https://twitter.com/davidjbland/status/467096015318036480

11.2: Product Operations

Another product concept relevant to an architecture practice is **Product Operations**. Melissa Perri and Denise Tilles provide an excellent overview of Product Operations in their book[4]. Perri and Tilles define Product Operations as **the discipline of helping your product management function scale well**, surrounding teams with all of the essential inputs to set strategy, prioritize, and streamline ways of working.

image by gerd altmann from pixabay

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

1. **Business Data and Insights**: This pillar focuses on the internal collection and analysis of data to create and monitor strategies. It helps leaders track progress regarding outcomes, reconciling research and development (R&D) spending with return on investment (ROI). It also integrates business metrics such as annual recurring revenue

[4] https://www.productoperations.com/

(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**.

11.2.1: 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**.

11.2.2: 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**.

11.2.3: 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.

11.2.4: Product Operations and Architecture Practice

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

In organizations with Product Operations teams, an architecture practice can **create powerful synergic relationships**. Like an architecture practice, Product Operations aim to maintain **efficiency and cohesion across departments**. Product Operations function like an **orchestra conductor** by effectively managing critical data, activities, and communications, ensuring that each team contributes optimally to a unified vision.

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

11.3: 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?*

12: Decision Intelligence in IT Architecture: Learning From Data, Social, and Managerial Fields

image by sasun bughdaryan from istock

IN THIS SECTION, YOU WILL: Learn the basics of decision intelligence, the discipline of turning information into

better actions, and its relevance for an IT architecture practice. **KEY POINTS:**

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

Decision intelligence is a discipline concerned with selecting between options. It combines the best of **applied data science**, **social science**, and **managerial science** into a unified field that helps people use data to improve their lives, businesses, and the world around them. **Cassie Kozyrkov**[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 chapter, I want to share some golden nuggets I've gleaned from her teachings and how I've used them in the wild world of IT. IT architects, like decision-making ninjas, face critical choices every day. Here's how they flex their decision intelligence:

- By **making decisions** (e.g., deciding which cloud provider and services to use when moving applications from a private data center to a public cloud).

[1] https://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/

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

Decision intelligence is the secret sauce of IT architects' work in every twist and turn.

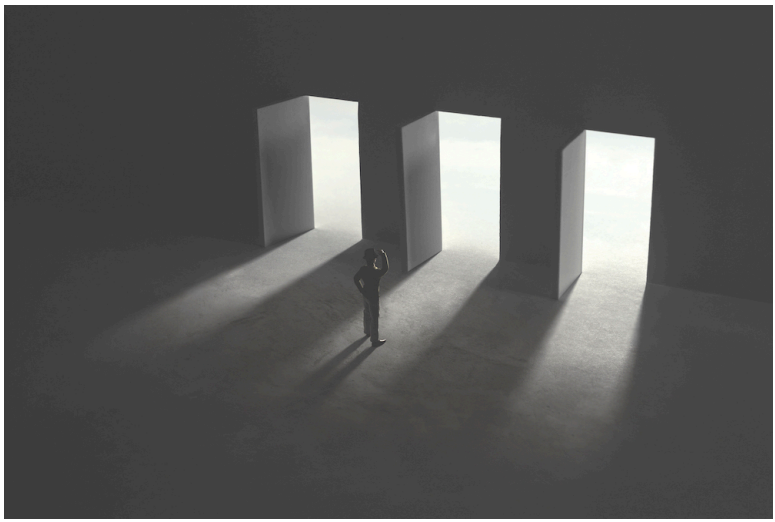

image by francescoch from istock

[4] https://martinfowler.com/articles/scaling-architecture-conversationally.html
[5] https://architectelevator.com/architecture/architecture-options/

12.1: Basics of Decision-Making

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

12.1.1: 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.

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.

[6] https://martinfowler.com/articles/oss-lockin.html

12.1.2: 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).

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

12.1.3: 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

[7] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9998299/
[8] https://en.wikipedia.org/wiki/Value_of_information

determine how much effort, info, and resources you should throw at a decision.

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

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

If you realize that perfect information isn't worth much for a particular decision, it's time to trust your gut. This strategy helps you balance the effort you put into decision-making with the decision's actual importance.

For example, deciding on the best public cloud provider is like choosing a life partner. This high-impact decision deserves thorough analysis. On the other hand, approving costs for an individual developer license that can be canceled at any time is like choosing what to have for lunch. Yet, companies often have procurement processes that make both these decisions feel like you're signing the Declaration of Independence.

So, next time you're stuck in a marathon meeting about whether to buy a €100 software library license, remember: not all decisions need to be treated like a royal decree. Save the deep dives for the big fish and keep the small fry simple!

12.2: 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.

12.2.1: Setting Goals

Practical goal setting is like trying to find your way out of an escape room—you need to understand your priorities and opportunities, or you'll run in circles. By identifying what's truly important and ignoring everyone else's distractions, you can focus on what really counts.

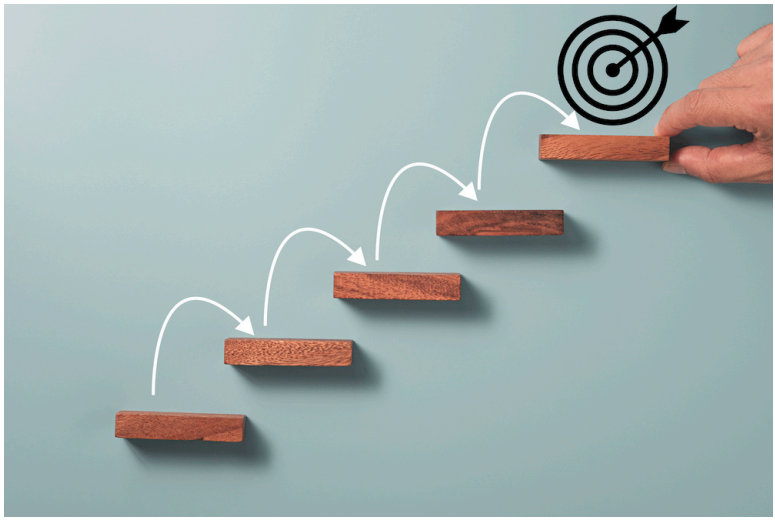

image by dilok klaisataporn from istock

Common goal-setting blunders include making goals so vague that they float away like a helium balloon or so rigid that they shatter at the first sign of trouble. The secret sauce? **Layered goals** that bring clarity, each serving a different purpose. In managerial sciences, goals come in three flavors: outcome, performance, and process goals.

- **Outcome Goals**: These are the grand finales, the ultimate wins, but they can be as vague as a politician's promise and influenced by things beyond your control. It's like "creating value for customers" and "being profitable." Nice, right? But how do you measure "value" and "profit" when you're busy fighting off existential crises?
- **Performance Goals**: These goals are measurable and, if you're not aiming for the moon, primarily within your control. In business, it's all about increasing website visits, slashing infrastructure costs, and boosting revenue. They're aspirational and tricky to manage, but hey, no pain, no gain.
- **Process Goals**: They're measurable and entirely within your control. In business, this translates to rolling out new features on time or launching a targeted marketing campaign. These goals keep you on track but should always serve your higher aspirations.

You need all three types of goals neatly connected like a well-oiled machine. For example, running a flashy marketing campaign (a process goal) should attract more visitors to your site and boost revenue (performance goals), ultimately delivering more value to customers and fattening your bottom line (outcome goals). Consolidating IT infrastructure (a process goal) should trim overall costs (a performance goal), making your business more profitable (an outcome goal).

But beware! Don't let process goals hog the spotlight and distract you from the big picture. Wise goal setting is all about layering your goals, aligning them with your priorities, setting ambitious targets, and establishing manageable processes, all while staying flexible and responsive to life's curveballs.

12.2.2: Aligning Goals: The Principal-Agent Problem

One of the classic headaches in goal setting for complex organizations is the **principal-agent problem**. This nifty concept from economics is like a plot twist in a soap opera: the interests of the decision-maker (the agent) are as different from those of the owner (the principal). For example, the owners (principals) may be about growth and expansion. At the same time, the managers (agents) might dream of longer lunch breaks and fatter paychecks. This **clash of interests** can lead to a mismanagement mess if mishandled.

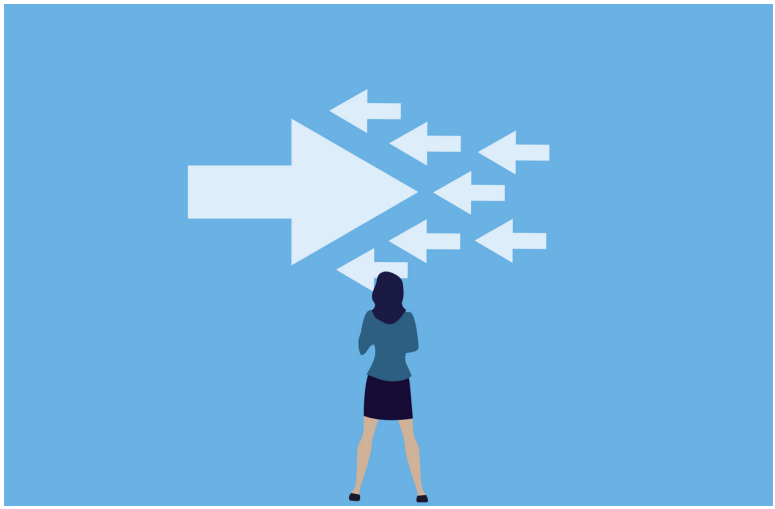

image by maria stavreva from istock

In the wild world of IT, a prime example is **technology selection**. Individual teams might want to use the latest, coolest tech based on their personal preferences. But letting each team run wild can turn your technology landscape into a tangled jungle. It's usually better for an IT organization to keep the tech menu limited. This way, it's easier to train newbies, maintain the codebase, and shuffle people between teams without causing a tech meltdown.

So, how do we get these misaligned interests on the same page? The principal needs to set up some **rules or constraints** to align the agent's decisions with their interests. This is like giving your dog a fenced yard—freedom to play, but within safe boundaries.

This principle also applies to personal decision-making, especially when juggling **short-term temptations** and **long-term goals**. By setting up **pre-emptive constraints**, you can steer your choices toward those long-term dreams and avoid decisions that might look tempting now but are as regrettable as a midnight snack of expired sushi.

For instance, in the technology selection saga, one strategy I often use is to create "golden paths[9]" —supporting a limited set of technologies and

[9] https://engineering.atspotify.com/2020/08/how-we-use-golden-paths-to-solve-fragmentation-in-our-software-ecosystem/

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

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

12.2.3: Is There A Decision To Be Made?

In decision-making, especially when you're not the one calling the shots, it's crucial to figure out how to contribute effectively as the decision whisperer. First things first: determine if there's even a space for a decision to be made. Sometimes, the big cheese has already decided and needs you to rubber-stamp it like a bureaucratic formality.

image by pheelings media from istock

Before diving into the murky waters of faux decision-making, clarify whether there's room for a decision. According to Cassie Kozyrkov, this involves two simple steps.

1. **Check the Decision-Maker's Auto-Pilot**: Figure out what the primary decision-maker would do if you weren't in the picture. Would they carry on like a self-driving car?
2. **Deploy the Magic Question**: Ask them, "**What would it take to change your mind?**" If they reply, "Nothing!" then congratulations, you've just discovered you're in a no-decision zone.

This latter question is your secret weapon for several reasons:

1. **Starts Insightful Conversations**: It's like opening a treasure chest of insights into the decision-making process and the decision-maker's mindset.
2. **Identifies the Decision-Maker**: It helps you figure out if the person you're talking to is the real decision-maker or just a messenger.
3. **Detects Real Decisions**: If no information can change their mind, then there's no real decision to be made. You're just there to wave pom-poms and cheer for a pre-made choice.

This magical question helps you map out the decision-making landscape, gauge the decision-maker's openness to new ideas, and see if there's genuine room for making or influencing a decision. If their mind is more closed than a locked vault, you'll know it's time to save your energy for real decision-making battles.

So, next time you're in a meeting and wondering if you're there to make a difference or to play the part of the wise advisor in a decision-making drama, remember to whip out the magic question. It's your ticket to knowing whether you're in for an epic decision-making saga or a cameo appearance.

12.3: Decision-Making Complexity

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

image by chaiyapruek2520 from istock

12.3.1: 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.

12.3.2: 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.

12.3.3: 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.

12.3.4: 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.

12.3.5: 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.

12.3.6: 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.

12.3.7: 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.

12.3.8: 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.

12.3.9: 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.

12.3.10: 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.

12.3.11: 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.

12.3.12: 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.

12.3.13: 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.

12.3.14: 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.

12.4: Decision is a Step in the Process

The Radical Candor book introduces a productivity-focused strategy[10] to improve decision-making processes, reduce time spent in unproductive meetings, and enhance overall efficiency. The central premise of this methodology is that you cannot just make a decision. To have any impact, you need to go through several steps: Listen, Clarify, Debate, Decide, Persuade, Implement, and Learn.

image by t_kimura from istock

12.4.1: Listen

The first step is to **Listen**. Gathering relevant data, feedback, and insights from various sources is crucial. Understanding different perspectives from stakeholders, team members, and experts helps form a comprehensive

[10] https://www.radicalcandor.com/wp-content/uploads/2022/09/How-To-Get-Shit-Done-With-Radical-Candor.pdf

view of the situation. This step ensures a thorough **understanding of the situation** by gathering relevant data and insights from various sources. It also ensures consideration of **diverse perspectives**, which helps form a well-rounded view of the issue.

12.4.2: Clarify

Next, you need to **Clarify**. Clearly articulate the issue or decision to ensure **everyone understands the context**. Establishing clear, specific goals and desired outcomes for the decision is essential. This sets the stage for focused discussions and aligned efforts. Clarifying the problem or decision you need to make ensures clear articulation and alignment on objectives and desired outcomes. This step **eliminates misunderstandings** and ambiguities, solidifying the foundation for focused discussions.

12.4.3: Debate

Following clarification, the step is to **Debate**. Engage in discussing potential solutions and alternatives, weighing the **pros and cons** of each option. Encourage an **open discussion** where team members can freely share their ideas and constructively challenge each other's viewpoints. This collaborative approach helps in exploring various aspects of the decision. Debating potential solutions ensures a thorough evaluation of all possible options. It promotes an open exchange of ideas and constructive challenges, allowing for exploring various aspects and implications of the decision.

12.4.4: Decide

Once the debate has provided a thorough evaluation of options, it's time to **Decide**. Choose the best course of action based on the gathered information and discussions. Depending on your organizational culture, you may reach a consensus among the team or have the decision made by a designated leader. The key is to make an **explicit and informed choice**. The decision-making step ensures an informed and thoughtful choice based on the available information and discussions. It provides a clear resolution and direction for moving forward, ensuring the best action is selected.

12.4.5: Persuade

After making the decision, the next step is to **Persuade**. Clearly **communicate** the decision and the rationale behind it to all stakeholders. It's essential to **gain buy-in** from team members and stakeholders by addressing their concerns and objections, convincing them of the decision's validity and importance. Persuading stakeholders and team members ensures effective communication of the decision and its rationale. This step secures buy-in and support, addressing and mitigating any concerns or objections that may arise.

12.4.6: Implement

With buy-in secured, you move to **Implement**. Execute the plan by **assigning tasks and responsibilities** to the appropriate individuals. Ensure everyone understands their role in the overall plan, providing the **necessary resources** and support to facilitate effective implementation. Alignment and clear communication are vital during this phase. Implementing the decision ensures the execution of the plan through the assignment of tasks and responsibilities. It ensures alignment and understanding of roles and the provision of necessary resources and support for successful implementation.

12.4.7: Learn

Finally, it's crucial to **Learn**. Monitor the implementation process and measure the outcomes against the set objectives. **Collect feedback** from stakeholders and team members to assess the effectiveness of the decision. **Analyze the results** to identify lessons learned and make necessary adjustments to improve future decision-making processes. This continuous improvement approach helps **refine strategies** and **enhance productivity** over time. Learning from the process ensures continuous improvement. Monitoring and measuring the implementation and outcomes, collecting and analyzing stakeholder feedback, and identifying lessons learned all refine future decision-making processes.

12.5: Decision-Making With Data and Tools

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

image by blue planet studio from istock

12.5.1: Remember that data has limitations

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

For instance, in artificial intelligence (AI), **AI biases stem from the data** it's fed, reflecting the choices and prejudices of those who compile the data. The issues with AI bias are often due to poor decisions regarding data selection. **Data isn't inherently objective**; it carries the **implicit values of its creators**.

Data's value lies in its ability to **enhance memory**, **not ensure objectivity**. Embracing data means embracing a significant advancement in human potential. It's about transforming information into action, extending beyond the limits of personal memory to make better, more informed decisions.

12.5.2: Chose the right tool for the job

Cassie Kozyrkov breaks down the techniques for analyzing data into three snazzy groups[11]:

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.

12.5.3: Prefer complete information to partial information

No matter how you plan to use data, full information is always preferable to partial information. If you only have partial information, you're dealing

[11] https://kozyrkov.medium.com/what-on-earth-is-data-science-eb1237d8cb37

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.

12.5.4: Be in the driving seat

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

As a decision-maker, it's crucial to **ask the right questions**, and figure out **which decisions are worth your brainpower**. Once you've identified those critical decisions, then—and only then—should you whip out the advanced statistical or other methods to get more accurate answers in the murky waters of uncertainty. Diving into data without asking the right questions is like wandering into a labyrinth with a blindfold on.

12.6: To Probe Further

- Introduction to Decision Intelligence[12], by Cassie Kozyrkov, 2020
- Decision Intelligence: The steering wheel for your life[13], by Cassie Kozyrkov, 2024
- Making decisions right, even if they're not always the right decision[14], by Gregor Hohpe, 2021
- Are IT's biggest decisions its worst?[15], by Gregor Hohpe, 2019
- Your most important architecture decisions might be the ones you didn't know you made[16], by Gregor Hohpe, 2020

[12] https://www.linkedin.com/pulse/introduction-decision-intelligence-cassie-kozyrkov/
[13] https://www.linkedin.com/learning/decision-intelligence/
[14] https://architectelevator.com/strategy/always-be-right/
[15] https://architectelevator.com/transformation/it-decisions/
[16] https://architectelevator.com/architecture/important-decisions/

12.7: Questions to Consider

- *How do you typically approach decision-making in your professional role, and in what ways could you incorporate the principles of decision intelligence to enhance your decision-making process?*
- *Have you observed instances where excessive organizational complexity resulted from poor decision-making? How can IT architects address this, and what role can they play in simplifying decision-making processes?*
- *Reflect on a recent significant decision you made. Were you aware of the resources you were committing and the opportunities you were preceding? How could you have evaluated these factors more effectively?*
- *Think of a situation where the outcome of a decision didn't align with your expectations. How did you judge the quality of the decision-making process in hindsight, and did you consider the role of luck or randomness?*
- *Consider a recent decision you faced. What would have been the value of perfect information in that scenario? How does this concept help you balance the effort and resources you allocate to different decisions?*
- *How do you set and align your goals, and what challenges have you faced? Are there instances where misalignment has led to ineffective decision-making?*
- *What factors have you found to increase the complexity of decision-making in your experience? How do you manage these complexities effectively?*
- *How do you use data in your decision-making process? Are there instances where data has misled your decisions, and how can you safeguard against this in the future?*

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

image by elxeneize from istock

IN THIS SECTION, YOU WILL: Learn that IT transformation projects face similar challenges as other mega-projects, often failing due to cognitive biases and poor planning, but applying key lessons from successful projects—such as risk mitigation, modular design, and stakeholder engagement—can significantly improve their outcomes.

KEY POINTS:

- IT transformation projects are large-scale, multi-year endeavors that often aim for significant organizational changes, such as creating unified IT platforms after mergers, and they frequently face challenges similar to other mega-projects.
- Common cognitive biases and decision-making fallacies like overconfidence, strategic misrepresentation, and the sunk cost fallacy often derail these projects, leading to cost overruns, delays, and underperformance.
- Flyvbjerg and Gardner's book "How Big Things Get Done" highlights lessons from successful and failed mega-projects across various industries, offering 11 practical heuristics for improving project leadership, including hiring the right experts, building modular systems, and focusing on risk mitigation.
- Key strategies for IT success include taking the outside view to learn from similar projects, fostering strong stakeholder relationships, and avoiding unnecessary complexity or scope creep by staying focused on core objectives.

Most **IT transformations** involve an immense, multi-year, multi-million-dollar project. Contrary to continuous improvements, IT transformation projects frequently aim to create significant organizational changes, such as creating a single IT platform after merging dozens of companies. Due to their size, IT transformation projects belong to the **category of mega projects**. They can benefit from lessons that other mega projects have learned.

For practical strategies on how to navigate these challenges, '**How Big Things Get Done**' by Bent Flyvbjerg and Dan Gardner is an invaluable resource. This comprehensive exploration of large-scale projects not only identifies why they often fall short of expectations but also provides actionable steps to improve their success rates. Drawing from extensive research, real-world case studies, and psychological insights, the book offers a practical roadmap for better planning and execution, equipping people with the tools they need to succeed.

Flyvbjerg and Gardner discuss how project planners and stakeholders are

often **overly optimistic** or intentionally misrepresent facts to win approval, leading to budget overruns and delays. Large projects like airports, bridges, and Olympic infrastructure are notorious for going over budget and time, just like many IT projects. Flyvbjerg and Gardner explore how human psychology affects decision-making and project outcomes. They underscore the importance of fighting cognitive biases and fostering a culture of honesty and transparency in project planning, making it clear that these values are not just desirable, but necessary for successful project outcomes.

The book is a treasure trove of examples from successful and failed projects across various industries, including technology, construction, and entertainment. Contrasting examples like the successful **Pixar movie-making process** with troubled infrastructure projects provides a comprehensive understanding of the factors that contribute to project success or failure, enlightening the reader and informing their future decisions.

13.1: The Iron Law of Mega-Projects

Flyvbjerg and Gardner define the iron law of mega-projects as follows: ***"Over budget, over time, under benefits, again and again."*** These words resonate with the **brutal reality** of mega-projects. Put simply, the typical project is one in which costs are underestimated and benefits are overestimated. Picture a big project that costs more than it was supposed to and delivers less than expected. Flyvbjerg and Gardner analyze famous examples, such as the **Sydney Opera House** (ten years late and 1,357% over budget[1]) and the **Big Dig in Boston**, (nine years late and 190% over budget[2]) explaining why these problems are so widespread.

Here's what the data shows from their study of 16,000 such projects:

- A mere 0.50% hit budget, time, and benefit expectations (or better)
- 8.50% meet both budget and time targets (or better)
- 47.90% manage to stay on budget (or better)

That is, sadly, typical. On project after project, **rushed**, **superficial planning** is followed by a **quick start** that makes everybody happy because shovels are in the ground. But inevitably, the project crashes into **problems that were overlooked** or not seriously analyzed and dealt with in planning. People run around trying to fix things. More stuff breaks. There is more running around. Flyvbjerg and Gardner call this the "**break-fix cycle**", like a "mammoth stuck in a tar pit." Several cognitive biases and decision-making fallacies often sabotage these efforts, leading to cost overruns, delays, and underperformance. All these factors together frequently lead to a negative learning loop (Figure 1): *"The more you learn, the more difficult and costly it gets."*

[1] https://en.wikipedia.org/wiki/Sydney_Opera_House#Completion_and_cost
[2] https://en.wikipedia.org/wiki/Big_Dig

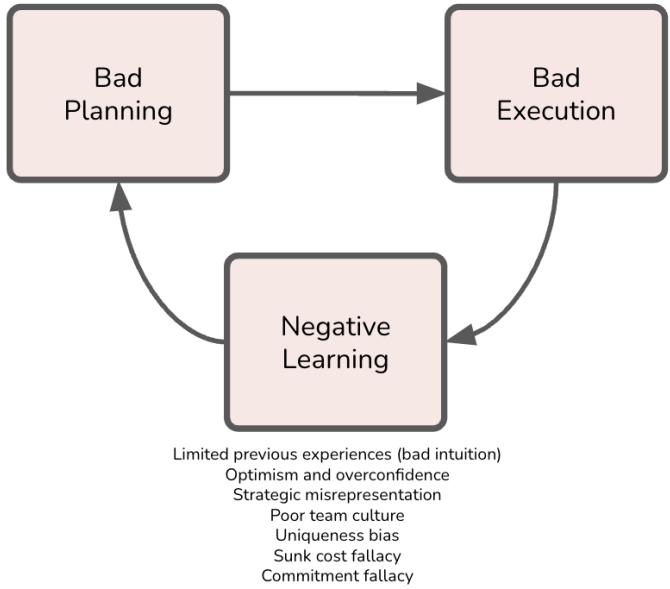

***Figure 1**: The negative learning loop behind many mega-projects.*

For IT projects, Flyvbjerg and Gardner gave the following conservative estimates of base rates for cost risks based on their database of projects:

- Mean cost of overrun of IT projects is 73%
- 18% of IT projects belong to the long tail (>= 50% of overrun costs)
- Mean cost of overrun of IT projects in the long tail is 447%

This analysis means that even if you reserve an extra 100% of your planned budget for unforeseen problems, you still have a fair chance of ending up in the long tail, where costs could be arbitrarily high.

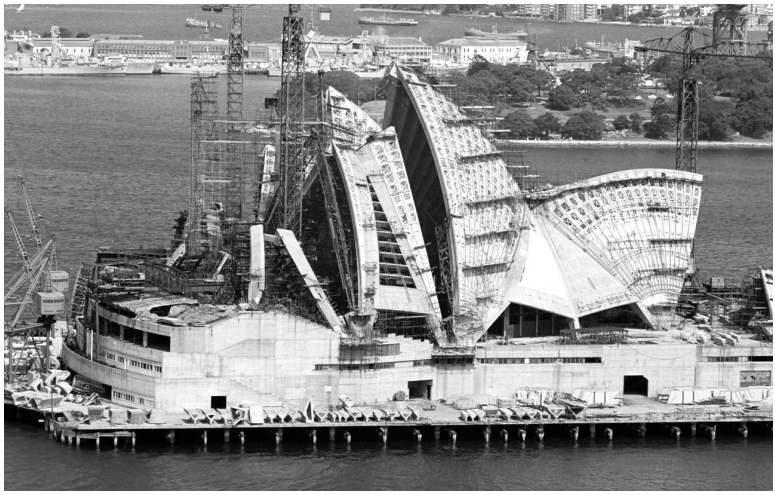

image by johncarnemolla from istock

Let's explore some of typical mega-project pitfalls:

- Optimism and Overconfidence (Hope Is Not a Strategy)
- Strategic Misrepresentation
- Bad Team
- Uniqueness Bias
- Lack of Experience (Eternal Beginner Syndrome)
- Sunk Cost and Commitment Fallacies

13.1.1: Optimism and Overconfidence

Leaders and stakeholders frequently **overestimate their ability** to deliver complex transformations. Flyvbjerg and Gardner's key heuristic for managing optimism on projects is *"You want the flight attendant, not the pilot, to be an optimist."*

image by skynesher from istock

It's common for optimism and overconfidence to rear their heads in IT projects. Leaders and stakeholders may find themselves underestimating the complexity and risks involved. Here are some examples that you might find familiar:

- **Cloud Migration**: A company migrating from on-premise servers to a cloud-based infrastructure may **overestimate the skills** of its engineering team, assuming they can handle the complexity without significant retraining. This assumption can lead to unexpected integration challenges, especially when dealing with legacy systems that were not designed for cloud environments. As a result, timelines get extended, budgets are exceeded, and operational disruptions occur during the migration process.
- **Microservices Refactoring**: A software engineering team may decide to refactor their monolithic application into a microservices architecture, believing their existing codebase is stable enough to handle the transition seamlessly. They might **overlook the technical debt** accumulated in the monolithic system, such as tightly coupled components and poor documentation. This overconfidence in the system's stability can lead to unforeseen challenges, like managing inter-service communication, increased latency, or scaling issues, which delay the project and inflate costs.

- **AI Deployment**: In a project implementing AI or machine learning algorithms, a team might be overly optimistic about their ability to develop, train, and deploy the models effectively. They may ignore less predictable and the **time-consuming** processes of **data cleaning** and **model validation**, hoping the models will perform well immediately. This optimism can lead to delays when the models fail to deliver the expected results in production, forcing the team to go back and address foundational issues.

In all these cases, **overconfidence** can lead to a failure to properly **assess risks** and plan for the necessary resources, while **optimism** creates **unrealistic expectations** about timelines and project success. By underestimating early delays, leaders also fail to account for how delays compound over time, ultimately jeopardizing the entire project.

An interesting form of overconfidence is that **early delays** are **not seen as a big deal** by most project leaders. They figure they have time to catch up because the delays happen early. That sounds reasonable. But it's dead wrong. Early delays **cause chain reactions** throughout the delivery process. The later a delay comes, the less work there is and the less the risk and impact of a chain reaction. For instance, if a database migration is delayed by a few weeks due to integration issues, project managers may downplay the impact, assuming future phases can compensate for the lost time. However, these early delays often create a chain reaction, pushing back testing, deployment, and training phases, resulting in even greater delays and increased costs down the line. Or, as US President Franklin Roosevelt said, *"Lost ground can always be regained—lost time never."*

13.1.2: Strategic Misrepresentation

In some cases, leaders **knowingly present overly optimistic forecasts** to get funding approval or to gain executive buy-in. This strategy **avoids rejection** from stakeholders who may be wary of the true timeline and costs.

image by catalin205 from istock

Here are some specific examples:

- **ERP System Overhaul**: A transformation project leader might present a rosy forecast for a large-scale Enterprise Resource Planning (ERP) system upgrade, promising that it will be completed within a year. Knowing that typical ERP implementations involve complex integrations with legacy systems, data migration, and extensive testing, the leader may hide these challenges to avoid scaring off executive stakeholders with a more realistic multi-year timeline. As the project progresses, the team quickly runs into unforeseen challenges, like data compatibility issues or workflow changes, which lead to delays, budget overruns, and frustration from the IT team and the executives who feel misled.
- **Software Refactoring**: A project manager might downplay the effort required to refactor a legacy system to make it more modular or scalable. They may minimize the technical debt associated with the old codebase—such as outdated libraries, hardcoded dependencies, or poor documentation—knowing that acknowledging these issues would require a larger budget and longer timeline. To secure immediate funding or developer resources, the manager assures leadership that the refactoring can be done quickly. However, as

engineers dive into the project, they discover that cleaning up the technical debt is far more complex than expected, leading to missed deadlines, increased costs, and team burnout.

- **Cloud Migration Misrepresentation**: A transformation project leader might promise that migrating a company's IT infrastructure to the cloud will result in immediate cost savings and improved efficiency within six months, downplaying the complexities of migrating legacy systems, rearchitecting applications for cloud-native services and addressing security concerns. Knowing that a more honest projection—perhaps 18 to 24 months—might reduce executive enthusiasm for the project, the CTO strategically presents an overly optimistic view. As the project unfolds, unexpected integration challenges arise, such as ensuring data consistency across cloud and on-prem environments, leading to delays and escalating costs that eventually sour stakeholders on the project.
- **Cybersecurity Overhaul**: An IT security leader might strategically underplay the scope of a cybersecurity overhaul, presenting the project as a relatively simple implementation of new tools, like firewalls and intrusion detection systems, while neglecting to mention the need for organization-wide security policy updates, employee training, and long-term monitoring. This underrepresentation might be used to secure quick buy-in from executives who are resistant to the idea of a larger-scale project. However, as the implementation begins, it becomes clear that these foundational changes are necessary, leading to scope creep, increased costs, and delays that frustrate both the project team and stakeholders.

In all of these cases, **strategic misrepresentation** might provide **short-term benefits** in the form of quick project approval and initial enthusiasm, but it sets the stage for **long-term failure**. Once the inevitable delays, cost overruns, and technical challenges surface, trust between leadership and the project team erodes, and the organization ends up paying a higher price—in both time and money—than if the project had been scoped and communicated accurately from the start.

13.1.3: Poor Team Culture

A strong, cohesive team is essential in mega-projects, where complexity and stakes are high. A **lousy team** not only **undermines performance**

but also magnifies risks, leading to **cascading failures** that jeopardize the entire project's success. In IT mega-projects a bad team can have severely negative impacts, amplifying the inherent complexities and risks.

image by charday penn from istock

When team members are disjointed, unmotivated, or lack trust in one another, several issues arise that can derail the project:

- **Poor Communication and Collaboration**: A lack of cohesion in the team leads to miscommunication, misunderstandings, and siloed efforts. This results in inefficient workflows, duplicated work, and critical information being missed or ignored, causing delays and costly rework.
- **Low Morale and Disengagement**: When team members do not feel empowered or connected to a shared goal, they become disengaged and less committed. This disengagement can lead to poor decision-making, suboptimal performance, and a lack of accountability, which in turn increases the likelihood of errors and project delays.
- **Inconsistent Quality and Execution**: A poorly structured team will struggle to maintain consistent work standards. Without a

unified approach, varying levels of expertise and different working styles can create gaps in quality, leading to technical debt, integration issues, and unreliable systems that require additional time and resources to fix.

- **Increased Conflict and Blame Culture**: When team members are not aligned, conflicts over priorities, responsibilities, and blame for failures arise. This misalignment further damages morale slows decision-making and distracts from the core objectives, exacerbating delays and overruns.
- **Failure to Adapt to Challenges**: A dysfunctional team is less likely to adapt to inevitable project challenges, such as scope changes or unexpected technical issues. Without trust and a shared commitment to problem-solving, the team may be slow to respond to setbacks, resulting in missed deadlines, increased costs, and project failure.

Here are specific examples in IT projects where a bad team can severely impact execution:

- **Cloud Migration Project**: In a large-scale cloud migration project, where multiple teams (networking, security, application development, and operations) must work together, poor communication can lead to critical gaps. For example, if the networking team does not correctly communicate firewall rules and IP whitelisting requirements to the operations team, the migrated applications might fail to connect to necessary resources, leading to significant downtime and troubleshooting. If different teams use inconsistent approaches for migrating workloads (e.g., one team optimizes for performance while another focuses on cost), the result can be a fragmented cloud architecture, with some applications running smoothly and others facing performance bottlenecks or security vulnerabilities.
- **Microservices Architecture Refactor**: In a project to refactor a monolithic application into microservices, team members may resist change due to the complexity and additional workload. If developers feel unsupported or disconnected from the project's vision, they may not fully commit to learning new frameworks or technologies required for microservices. This situation can result in poorly designed services with limited scalability and high maintenance costs. Microservices require teams to adopt DevOps practices

like continuous integration/continuous deployment (CI/CD). If the team is dysfunctional and does not fully embrace these practices, deployments may become manual and error-prone, leading to frequent downtime and slower release cycles.

- **Data Warehouse Implementation**: Conflicts can arise between data teams in a data warehouse project that consolidates data from multiple departments. If there is no alignment on data governance and standards, teams might blame each other for data inconsistencies or performance issues. For instance, the finance team may report issues with slow queries. In contrast, the engineering team insists that the database schema is not the problem, resulting in a standoff and no resolution. Different teams may load data into the warehouse with inconsistent validation checks. This inconsistency leads to corrupted or duplicate data, causing critical business reports to be inaccurate. Fixing these data quality issues after deployment would require additional resources, delays, and erode stakeholder trust in the system.
- **Agile Transformation for Software Development**: In an Agile transformation project, if teams do not trust or fully adopt Agile principles, such as frequent feedback loops and iterative development, they may revert to old habits of waterfall planning. This results in poor responsiveness to changing business needs missed sprint goals, and an inability to deliver incremental value. Agile ceremonies like sprint reviews or retrospectives without cross-functional collaboration become box-ticking exercises rather than meaningful ways to improve the process. Agile transformations require a shift in mindset. However, team members may become disengaged if the team feels that management is only paying lip service to the principles without making fundamental organizational changes (like reducing micromanagement). Disengaged teams miss opportunities to optimize workflows, introduce better tools, and improve velocity, ultimately stalling the transformation effort.
- **Custom CRM System Development**: In a project to build a custom CRM system for a large organization, poor collaboration between frontend and backend teams can result in misaligned system design. For example, the backend team might design inefficient or difficult APIs for the frontend team to use, causing delays in integration and forcing the frontend team to write additional code to make the system work as intended. If the development team lacks a unified

testing strategy, some features may be well-tested while others contain bugs or performance issues that go unnoticed until production. This lack of strategy leads to frequent bug fixes post-launch, which frustrates users and increases long-term maintenance costs.

- **Cybersecurity Implementation in a Financial Institution**: In a cybersecurity overhaul, if the security team is not aligned with the development and operations teams, each group may prioritize its own tasks over shared security goals. For example, the development team might push out new features without proper security reviews, blaming the security team for delays, while the security team blames developers for introducing vulnerabilities. This conflict creates gaps in the system, exposing the organization to potential data breaches or compliance failures.

In these examples, a **lousy team dynamic**, characterized by poor communication, lack of engagement, inconsistency, and conflict, amplifies risks and delays in IT mega-projects. These issues affect technical outcomes and strain budgets, timelines, and stakeholder confidence.

13.1.4: Uniqueness Bias

Uniqueness bias means people see their projects as unique, one-off ventures with **little or nothing to learn from earlier projects**. If you imagine that your project is so different from other projects that you have nothing to learn from them, you will **overlook risks** you would catch and mitigate if you switched to the outside view instead.

image by yuri_arcurs from istock

IT transformations are often seen as unique by those executing them, leading to a reluctance to learn from other industries or similar projects. However, the reality is that lessons from other sectors, such as construction or film production, can be valuable. Here are some examples of how this bias plays out and its consequences:

- **Custom Enterprise Solution Development**: A company developing a custom enterprise resource planning (ERP) solution might believe its specific business processes are so unique that it cannot use existing frameworks or off-the-shelf solutions. As a result, the company might build the entire solution from scratch rather than adopting and customizing proven ERP systems. This "reinventing-the-wheel" approach leads to unnecessary complexity and development delays. Established ERP systems often have pre-built modules for common business processes (finance, HR, supply chain), and trying to recreate these from scratch introduces risks of inefficiencies, bugs, and a longer time to market. By failing to learn from existing systems, the company overlooks best practices in data management, integration, and compliance, increasing the risk of costly rework or system failure.
- **Cloud Migration Project**: During a cloud migration, IT leaders might believe that their company's legacy systems and data architecture are so unique that they require a completely custom

approach to the cloud strategy. This view leads them to ignore established cloud migration frameworks or case studies from other industries that have faced similar challenges. By not learning from similar cloud migrations, the team may overlook common risks, such as data migration errors, service downtime, or security vulnerabilities. Established frameworks, like those provided by cloud service providers (AWS, Azure), offer proven processes for phased migration, testing, and scalability. Still, a team with a uniqueness bias might dismiss these, leading to project delays and potential operational disruptions.

- **DevOps Implementation**: A software engineering team tasked with implementing a DevOps pipeline might believe their existing software development environment is so specialized that common DevOps tools and practices won't work. They might create custom deployment and testing pipelines, ignoring widely adopted tools like Jenkins, Docker, or Kubernetes. Building custom solutions increases the project's complexity and adds long-term maintenance costs. Commonly used DevOps tools are battle-tested, scalable, and come with extensive community support. By rejecting these tools, the team risks developing systems that are less reliable, more challenging to scale, and more costly to maintain, all of which could have been avoided by adopting industry-standard solutions.
- **Security Overhaul in a Financial Institution**: A financial institution undergoing a cybersecurity overhaul might view its security needs as so unique due to regulatory requirements that it avoids using best practices or tools that have been widely adopted in the industry. Instead, they may build custom security protocols or ignore lessons learned from other institutions' security breaches. This decision could lead to increased vulnerability to attacks or compliance failures. Lessons from other industries, such as incident response plans, network segmentation, or automated threat detection, would provide a robust framework for addressing risks. By failing to learn from others, the institution might introduce avoidable weaknesses into its security system, making it more prone to breaches or compliance penalties.
- **Overlooking Project Management Practices**: A large-scale software engineering project might view itself as fundamentally different from other industries, such as construction or manufacturing, dismissing project management practices commonly used in those

sectors. Teams might underestimate the value of phased delivery, risk management frameworks, or using a project's "critical path" to prioritize tasks. Ignoring lessons from other fields, the team fails to implement effective risk mitigation strategies, leading to unexpected scope creep, budget overruns, and delays. In contrast, large infrastructure projects often use planning techniques that account for risks and dependencies, allowing for better control of timelines and costs. Adopting similar methodologies could help IT teams manage complexity and reduce uncertainty.

In each of these cases, **uniqueness bias** prevents teams from benefiting from proven strategies and lessons learned in similar or even unrelated fields. By adopting the **outside view** and learning from both similar IT projects and other industries, teams can avoid common pitfalls, reduce risk, and improve project success rates.

13.1.5: Lack of Experience (Eternal Beginner Syndrome)

When an organization embarks on its first large-scale IT transformation, it often lacks experience in managing complex systems and technologies. This **lack of institutional memory** leads to **repeated mistakes** that more experienced teams could have avoided. For instance, in an IT architecture overhaul, the team might struggle to implement a scalable infrastructure due to poor planning or inadequate tools despite established best practices in the industry. In software engineering, teams might make the same coding mistakes, like inadequate version control or failure to manage dependencies properly, because they lack a disciplined software development lifecycle (SDLC).

image by skynesher from istock

Here are some examples illustrating this issue:

- **IT Architecture Overhaul**: A company undergoing its first IT architecture overhaul to implement a microservices-based architecture may lack experience in designing scalable systems. The team might underestimate the complexity of decoupling tightly integrated legacy systems, failing to plan for key elements such as inter-service communication, data consistency, and fault tolerance. Without adequate planning and knowledge of best practices, the company may end up with a fragile architecture that suffers from frequent outages or performance bottlenecks. Experienced teams would have anticipated the need for load balancing, service discovery, and failure recovery mechanisms, but the lack of experience leads to significant downtime and costly rework to stabilize the system.
- **Data Migration to the Cloud**: A company with no prior experience in cloud migration may attempt to move its entire on-premise data infrastructure to a cloud provider without fully understanding the implications of bandwidth constraints, data security requirements, or cloud-native application design. They might fail to implement proper backup strategies or to account for the potential latency

issues with large datasets. The lack of experience leads to failed migrations, with critical data being lost or corrupted in the process. Additionally, unoptimized applications and data structures cause excessive cloud costs and slow performance. These issues could have been mitigated with a more experienced approach, such as phased migration, cost management tools, or cloud optimization practices.

- **DevOps and CI/CD Implementation**: An organization attempting to adopt DevOps practices for the first time might struggle to establish continuous integration (CI) and continuous deployment (CD) pipelines due to a lack of familiarity with the necessary tools and automation. They may fail to properly set up test automation, monitoring, or infrastructure as code (IaC). Without a clear understanding of DevOps principles, the team ends up with manual deployment processes prone to human error, resulting in slower releases, frequent production issues, and high maintenance costs. Experienced teams would have prioritized automating the build, test, and deployment processes, ensuring more reliable and faster iterations.
- **Security and Compliance in IT Systems**: A company handling sensitive customer data for the first time might lack the necessary experience to implement proper security and compliance protocols, such as encryption, role-based access control, or regular security audits. The lack of institutional knowledge on data protection leads to significant vulnerabilities, exposing the organization to data breaches or non-compliance with regulations like GDPR or HIPAA. Experienced teams would have implemented security best practices from the outset, ensuring that data is protected and compliance requirements are met.
- **Technical Debt Accumulation**: Inexperienced teams might focus on short-term goals and quick feature releases without considering long-term scalability or maintainability. As a result, they accumulate significant technical debt with complex, poorly documented code that is difficult to maintain or extend. This technical debt slows down future development and increases the likelihood of bugs and system crashes, as the underlying infrastructure is not built to handle new demands. More experienced teams would have balanced feature development with addressing technical debt, ensuring that the codebase remains maintainable and scalable.

In these examples, **Lack of Experience and Eternal Beginner Syndrome** leads to costly mistakes and delays. Organizations embarking on their first IT transformation often overlook best practices and proven methodologies, resulting in inefficiencies, poor performance, and technical failures. Organizations can avoid these repeated mistakes by learning from more experienced teams or consulting external experts and achieving more successful outcomes.

13.1.6: Sunk Cost and Commitment Fallacy

The **Sunk Cost Fallacy** and **Commitment Fallacy** are closely related cognitive biases that lead decision-makers to continue investing in a project, often well past the point of viability, to justify previous investments or out of reluctance to admit failure. Together, they create a cycle of sustained investment in projects that may be outdated, inefficient, or otherwise unworkable.

image by georgeclerk from istock

The **Sunk Cost Fallacy** occurs when organizations persist in allocating resources—time, money, or effort—based on what has already been invested rather than the project's current and future benefits. Meanwhile,

the **Commitment Fallacy** or **Commitment Bias** arises when decision-makers feel compelled to push forward with a failing initiative, fearing that abandoning it would mean admitting defeat or wasting prior commitments. This push often results in quick "lock-in" decisions that overlook alternatives and elevate project risk.

Here are examples of how these biases manifest:

- **Custom Data Warehouse vs. Cloud Solutions**: A company may continue investing in a costly, on-premises data warehouse, even when cloud-based solutions offer better performance and scalability. The justification lies in the large initial investments in infrastructure, which overshadow the inefficiency and higher maintenance costs compared to cloud alternatives. Instead of shifting to a cloud solution, the company experiences mounting operational inefficiencies.
- **Legacy Codebase in Obsolete Languages**: A software team may continue enhancing an outdated codebase due to the significant resources already spent on development and training. Despite modern frameworks offering greater ease of maintenance and scalability, the team resists change, ultimately leading to a bloated, hard-to-maintain system that falls behind in performance.
- **Over-customized ERP Systems**: Companies that heavily customize ERP systems to fit their needs face issues when updates or integrations become difficult. Despite the challenges and escalating costs, they continue investing in these customizations due to prior investments, ultimately impeding the company's flexibility and competitiveness compared to more adaptable ERP options.
- **In-house Software vs. Off-the-Shelf Solutions**: Companies often invest in building custom solutions when off-the-shelf options are available. Even as maintenance becomes costly and new features are slow to implement, they avoid switching because of prior development costs, risking inefficiency and missed opportunities to streamline operations.

These fallacies lead to several detrimental impacts:

- **Inefficient Resource Use**: Continued investment in outdated or failing projects wastes valuable resources.

- **Increased Project Risk**: Projects suffer from technical debt, rigidity, and escalating costs that increase long-term risks.
- **Missed Opportunities**: By resisting modern, flexible solutions, companies miss out on enhanced scalability, cost savings, and competitive advantage.

To avoid these biases, organizations should adopt a data-driven approach that includes regular, objective reassessments of projects based on current value and future potential, not past investments. This approach can enable timely pivots to alternative solutions, reducing long-term costs and fostering innovation. Recognizing when to re-evaluate, abandon, or adapt projects is essential for sustainable growth and competitiveness.

13.2: Heuristics for Successful Mega-Projects

Cognitive biases and fallacies represent a significant risk in IT transformations. Overcoming them requires **rigorous active planning**, the **willingness to learn** from other industries and past mistakes, and a focus on making data-driven decisions. Successful IT transformations often involve **transparency**, **adaptability**, and a **willingness to pivot** when necessary rather than sticking to flawed assumptions, unrealistic optimism, or sunk investments.

image by skynesher from istock

Flyvbjerg and Gardner do not only analyze projects but have identified 11 heuristics for better project leadership:

- Hire a Masterbuilder
- Get Your Team Right
- Ask Why
- Build With Lego
- Think Slow, Act Fast

- Take The Outside View
- Make Friends and Keep Them Friendly
- Watch Your Downside
- Know That Your Biggest Risk Is You
- Say No and Walk Away
- Build Climate Mitigation Into Your Project

These techniques can set a project on the right path, creating a positive learning loop that pushes projects forward and improves over time.

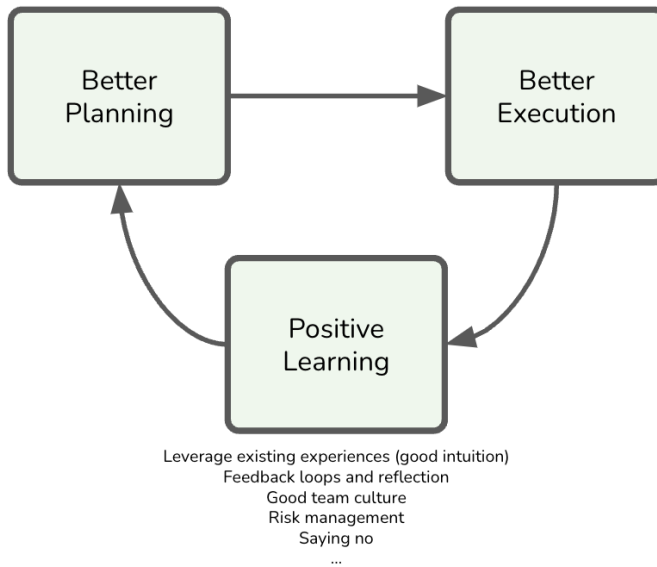

Figure 2: *The positive learning loop.*

13.2.1: Hire Experienced Masterbuilders

In any complex IT project, whether designing a new system, migrating to the cloud, or building a software platform, having **exemplary leadership** can make or break the effort. Flyvbjerg and Gardner claim that hiring a **master builder**—with deep expertise and hands-on experience—should be

your top priority. The term originates from the skilled masons who built Europe's medieval cathedrals, individuals with both the vision and the practical knowledge to bring massive, intricate projects to life. In today's world, it means finding someone with a **track record of successfully leading** similar IT initiatives.

It is important to understand what Flyvbjerg and Gardner mean by a master builder. I interpret their heuristics for a master builder as someone who **isn't necessarily controlling every detail upfront** but rather someone who has relevant experience and can **lead change effectively**, ensuring the project evolves successfully in real-time while balancing both strategic and operational priorities. A master builder is not necessarily a top-down architect who rigidly controls every aspect of a project. Instead, a master builder can be any key figure who knows how to drive change in an unpredictable, complex environment. This role is less about dictating every decision and more about **enabling the project to adapt and succeed** amidst constant flux, instilling confidence in the team and stakeholders.

In IT projects, a master builder could be a **Chief Technology Officer (CTO)**, **Chief Product Officer (CTO)**, a **Chief Architect**, a Senior Product Manager (or a team of them)—any senior leader with strong, relevant experience, the deep expertise and strategic vision required to steer the team through unforeseen challenges. They don't merely impose a rigid vision; they understand the technical and business intricacies, foresee potential risks, and know how to pivot when things don't go as planned. Essentially, they act as adaptive leaders who empower their teams and navigate uncertainty, keeping the project aligned with its goals while responding to emerging challenges.

image by freshsplash from istock

For instance, if you're overseeing a major system integration, your master builder might be a **seasoned CTO**, **chief architect**, **or principal engineer** who's navigated the complexities of combining legacy systems with modern cloud services. They'll know the potential pitfalls—like data migration challenges or API compatibility issues—and how to avoid them. Their domain experience allows them to **anticipate risks**, adapt to changes, and keep the project on track.

But, as you may have experienced, master builders **aren't always available or affordable**, especially for smaller projects or organizations with limited budgets. If that's the case, you'll need to think creatively. You might look for someone with solid expertise but lacks the full experience of a master builder. In this case, you'll want to **build a strong support network** around them, such as hiring subject matter experts to supplement their knowledge or bringing in external consultants for critical phases of the project.

Another option is to foster internal talent by **investing in mentorship and training**. With the proper guidance, you may have **rising stars** on your team who can step into a master builder role. Over time, you build the expertise you need internally by allowing them to lead smaller projects or shadow more experienced colleagues.

In any case, whether you hire an established expert or grow your own, the key is to ensure that someone with deep domain knowledge is at the helm of your project. The master builder has the practical wisdom to ensure your project's success, guiding it through the technical and managerial challenges that inevitably arise.

13.2.2: Get Your Team Right

This heuristic is the one that every successful project leader will tell you is **non-negotiable**: *Get your team right.* As Ed Catmull, co-founder of Pixar, wisely said, ***"Give a good idea to a mediocre team, and they will screw it up. Give a mediocre idea to a great team, and they will either fix it or come up with something better. If you get the team right, chances are they will get the ideas right."*** This statement holds especially true in IT, where complex systems, evolving technologies, and tight deadlines make teamwork essential to success.

image by simonkr from istock

But who should assemble the team? Ideally, this should be the responsibility of a *master builder*. Picking the right team is arguably the master builder's most important job. Whether you're building a new cloud

infrastructure, implementing a security overhaul, or leading a software development project, the right team can overcome technical obstacles, innovate on the fly, and adapt to changing requirements.

In IT projects, the master builder's role is far from solitary. It's their deep expertise and **leadership that sets the tone**, but it's the **team that delivers the project**. For example, when designing a distributed architecture, the master builder would bring together the right mix of cloud architects, developers, DevOps engineers, and security experts, ensuring that someone fills each role with the technical skills and collaborative mindset to succeed. A well-assembled team has the technical chops and the ability to work together smoothly, troubleshoot issues, and stay focused on the project's overarching goals.

Let's take an example in software engineering: You may have a brilliant principal engineer (the master builder) who can architect an elegant solution. However, if the team around them lacks the necessary talent in areas like front-end development, API integration, or QA testing, the project is likely to fall short. The master builder's real genius lies in their technical knowledge and ability to pick, mentor, and manage a team that works together cohesively toward the project's success.

What if a master builder isn't available? In that case, the project manager needs to **focus on team composition**. They must evaluate not just each member's technical skills but also how well those individuals collaborate. In IT, a team that communicates well and adapts to new challenges will far outperform one technically brilliant but siloed or resistant to change.

Remember, teams, not individuals, deliver projects. So, to amend the earlier advice: *When possible, hire a master builder—and the master builder's team.* Their combined expertise, working in sync, will ultimately drive your project to success. Even when faced with a less-than-perfect plan, a great team can adapt, innovate, and deliver far beyond expectations.

13.2.3: Ask Why

Asking why you're doing your project is one of the most powerful ways to **stay focused on what really matters**—your ultimate purpose and the desired result. IT projects often get bogged down in technical details, changing requirements, or unforeseen obstacles, constantly returning to the core *"Why?"* keeps you **grounded in the bigger picture**. This

overarching goal should be placed in the metaphorical "box on the right" of your project plan—a constant reminder of the result you're striving for.

image by courtneyk from istock

For example, if you're leading a project to migrate systems to the cloud, the ultimate purpose is improving scalability and reducing operational costs. As the project progresses and technical challenges arise—such as data migration delays, compatibility issues, or security concerns—it's easy to get lost in the details. But a strong leader continuously asks, *"Why are we doing this?"* If your actions, resources, or decisions don't contribute directly to achieving the project's purpose, they need to be reconsidered.

For instance, when implementing a microservices architecture, getting caught up in the technical nuances of service design, API management, or deployment pipelines is easy. However, if the ultimate goal is improving system flexibility and development speed, you must evaluate every technical decision against that goal. This strategic evaluation ensures that every technical decision is goal-oriented and contributes directly to the desired outcome.

Asking "Why?" It can also help you **prioritize features** and **manage scope creep**. As your project evolves, stakeholders may request additional features or changes. Before agreeing to any modification, ask, "Why is this

necessary?" And "Does this change help achieve the primary goal?" If the answer is no, it may be best to push back or deprioritize the request in favor of what drives the project toward success. This approach gives you a sense of control and decisiveness in managing the project's scope.

By continuously asking *"Why?"* throughout the life of your project, you ensure that every decision and every action is contributing to the ultimate result. This helps you stay focused on delivering value and prevents you from getting sidetracked by details that don't serve the bigger picture.

13.2.4: Build With Lego

Big is best built from small. Just like how a towering wedding cake is essentially a stack of smaller cakes, many large-scale projects in IT and engineering are constructed from simple, repeatable building blocks. This modular approach allows you to **scale systems efficiently**, getting **better**, **faster**, **and cheaper as you grow**. Think of each small unit as a **Lego brick**—a fundamental building block that, when replicated, creates something far more powerful.

image by albertpego from istock

In IT, this principle is at the core of many successful architectures. For instance, consider **server farms**: the server is the Lego brick. A data center isn't a massive, monolithic structure; it's a collection of individual servers stacked and connected in ways that allow them to scale up or down as needed. Similarly, in cloud computing, **virtual machines** or **containers** act as Lego bricks, providing modular, repeatable computation units that can be easily replicated or distributed across a global network.

The same idea applies to **microservices architecture** in software development. Instead of building one massive, monolithic application, you create a series of small, independent services that communicate with each other. Each microservice is a Lego brick—focused, self-contained, and scalable. If one service needs to handle more traffic, you can scale it independently without affecting the rest of the system. This modular approach makes your application more flexible, maintainable, and easier to scale.

This idea isn't limited to software and server architecture. Take **infrastructure as code (IaC)**—where you define your IT infrastructure using code templates, automating the setup and deployment of systems. Each piece of infrastructure—a network configuration, a virtual machine, or a storage unit—can be considered a Lego brick. With IaC, you can deploy, replicate, and scale your infrastructure quickly and consistently by reusing and stacking these blocks.

Modularity also applies to **software development processes**. Agile methodologies break down considerable development efforts into small, manageable sprints, where teams build, test, and release incremental pieces of functionality. Each sprint is like adding another Lego brick to your larger product, allowing continual improvement and flexibility.

This approach has been successfully applied beyond IT—whether in solar farms (where each **solar panel** is a building block), in shipping (where **containers** are standardized units), or even in **education** (where modular learning systems allow customization and scalability). Its applicability is only limited by imagination.

Building your systems, software, or infrastructure with Lego-like components creates the flexibility and scalability needed for long-term growth. Each piece is small, manageable, and optimized independently, but they form something much more significant together. The beauty of this approach is that it's not only scalable but adaptable to changes and innovations, allowing you to improve incrementally as your needs evolve.

13.2.5: Think Slow, Act Fast

Taking the time to **plan thoroughly** is crucial because the stakes during execution are much higher. As the saying goes, "measure twice, cut once." What's the worst that can happen during planning? You may lose some code snippets or adjust the project roadmap. But what's the worst that can occur during delivery? Your data migration corrupts critical information, causing system-wide failures. Your untested API causes a security breach—your cloud infrastructure crashes during peak demand, costing millions in downtime. Almost any nightmare scenario can happen—and has happened—during delivery. That's why it's essential to limit your risk exposure.

You do this by **thinking slowly** during the active planning phase. In this stage, the **cost of reworking** a system design or rethinking your technology choices is **relatively low**. You can test ideas, simulate failures, and anticipate potential issues with little consequence. Thinking through every detail—designing the architecture for a new platform, setting up cloud infrastructure, or defining security protocols—pays off massively when it's time for action.

image by nanostockk from istock

Again, it is essential to understand what Flyvbjerg and Gardner mean by thinking slowly. Planning and thinking slowly is neither a theoretical

exercise nor a traditional waterfall approach where you decide everything up front without room for adjustment. Instead, it's a practical, iterative, and dynamic process that involves experimenting, simulating, exploring, and doing everything necessary to clarify your roadmap, reduce risks, and minimize surprises. In IT architecture and software engineering, this means testing different approaches, running simulations to expose potential failures, and experimenting with configurations to see what works best. The goal is to refine your strategy and eliminate uncertainties before you hit the high-stakes phase of full-scale delivery.

Unlike rigid waterfall planning, this approach encourages continuous feedback, real-time learning, and adaptability. For example, in planning a system migration or platform launch, you're not just brainstorming once and sticking to a static plan; you're actively testing workloads, simulating failure scenarios, and exploring alternative solutions as new information becomes available. This experimentation gives you a clear understanding of what to expect, where the risks are, and how to navigate them before you commit to the real-world rollout.

This approach ensures you're not just "thinking" slow—you're adjusting and learning to avoid the costly surprises that can derail a project during full-scale delivery. It's an approach that balances thoughtful planning with the flexibility to adapt.

Flyvbjerg and Gardner use Pixar's planning process as an example of an exemplary process for mega-projects. It is well-known for being highly iterative, collaborative, and flexible, allowing the team to explore ideas in depth before settling on a final narrative or visual direction. Here's a breakdown of how it works:

1. **Brainstorming and Concept Development**: Every project begins with a core idea, often a story or theme pitched by a director or team member. From here, Pixar holds brainstorming sessions to flesh out the concept, characters, and plot possibilities.
2. **Storyboarding and Story Reels**: Pixar artists create rough storyboards to visualize scenes. The artist then assembles storyboards into "story reels," like animated storyboards that give a sense of pacing and flow. This early visualization helps the team assess how scenes work together and identify potential issues with the story structure.
3. **Brain Trust Reviews**: One of Pixar's unique processes is the "Brain Trust." This trust is a group of experienced directors, writers, and

creatives who meet periodically to review the story's progress. They offer feedback, critique, and insights to help the director improve the story. The Brain Trust isn't there to dictate; instead, it provides unfiltered feedback, allowing the director to decide how to move forward.

4. **Iteration and Refinement**: Based on feedback, the team revisits storyboards and story reels, constantly refining and reworking scenes. Pixar embraces this iterative approach, repeatedly rethinking and reshaping the story until it resonates emotionally and makes narrative sense.
5. **Early Animation Tests and Visual Development**: Pixar experiments with rough animation and visual styles early on to see how characters will look and move. They may conduct tests with lighting, colors, and textures, exploring different visual approaches that align with the story's tone.
6. **Technical Development and Simulations**: Pixar's stories often require innovative animation techniques, so the technical team is involved early to develop tools, simulate effects, and experiment with new technologies. This step is akin to running simulations in software development to ensure their tools and techniques will work before total production.
7. **Pre-Production and Production**: Once the story is solidified, Pixar moves into pre-production, where it finalizes designs, builds models, and rigs characters. By this point, Pixar is confident in the story and its direction, having refined it through constant testing, feedback, and adjustment. The project then moves into full production, creating the final animations.
8. **Continued Feedback and Tweaks**: Even during production, Pixar maintains flexibility. They continue to screen the movie to the Brain Trust and even test audiences, using this feedback to make necessary adjustments. They're open to late-stage tweaks, ensuring the final product is polished and compelling.

This iterative planning and production model has been integral to Pixar's success. It allows them to refine their ideas meticulously and collaboratively while being open to creative shifts throughout the process. It balances deep, thoughtful exploration with a flexible approach that adapts to new insights and discoveries along the way.

For example, suppose you're planning a migration from on-premise servers to the cloud. In that case, meticulous planning can help you identify which workloads should move first, how to maintain system continuity, and how to test for issues before going live. Spending extra time to plan and test may seem slow, but it's a **fraction of the cost and hassle** compared to dealing with unanticipated failures during delivery. It can also reduce the chances of needing a complete rollback or emergency patch, which can be expensive and damaging to your reputation.

You **act fast** once the planning is complete and the **team understands the roadmap**. Delivery is where the real risks lie. Every decision made during delivery carries higher stakes—downtime, security breaches, data loss, or even failure to meet critical deadlines. The key is minimizing the window in which things can go wrong by executing precisely and quickly. In a software deployment, this means having automated testing and CI/CD pipelines in place so that when you act, you can quickly detect and fix issues before they spiral out of control.

Consider launching a new app or major system upgrade. During the planning phase, you've mapped out user journeys, tested scalability, and ensured the system can handle peak loads. Once you go live, acting fast means monitoring performance in real time, quickly resolving bottlenecks, and rolling out patches swiftly if issues arise. You want the period of uncertainty—when the system is most vulnerable—to be as short as possible.

Good planning boosts the odds of a quick, effective delivery. It allows you to act decisively, execute smoothly, and close the window on risk as fast as possible. Thinking slow and acting fast is how you control complex projects, limit the dangers, and ensure a successful outcome.

13.2.6: Take The Outside View

Your project may **feel unique and special**, but it is likely **part of a larger class of projects** unless you are working on something that has never been done before—like developing an AI that achieves accurate general intelligence or creating a quantum computing network. Getting caught up in the specifics of your application or system can be easy. However, thinking of your project as "**one of those** " (for example, one of many cloud migrations, microservices refactor, or platform integrations) will

allow you to gather data and learn from the collective experience of similar initiatives.

image by eoneren from istock

Reference-class forecasting is a powerful tool here. For instance, if you're planning a cloud migration, you can look at data from other companies or projects that have done the same thing. How long did it take them? What common issues did they face, like underestimating data transfer times or managing service disruptions? By using this data, you'll make more accurate predictions about your own project's timeline and resource needs.

The same applies when adopting a new technology stack. Say your team is switching from a monolithic architecture to microservices. While this might feel like a radical shift, there's a wealth of data from others who've made this transition. You can learn from the time it typically takes to decompose services, the common pitfalls like latency introduced by inter-service communication, and the operational challenges of managing a distributed system.

Risk mitigation also benefits from this outside view. For example, if you're implementing an API-first strategy, it's easy to focus on the specific requirements of your API consumers. But take the outside view and look at other API-first projects. You might discover risk patterns—such

as API versioning issues, backward compatibility, or security concerns—that many projects face. Recognizing these risks early on, because they are common across projects of this nature, will help you address them proactively.

By shifting your focus from your project's uniqueness to the class of projects it belongs to, you'll have a **more accurate understanding of what to expect**. The broader perspective will allow you to make better-informed decisions about timelines, costs, and risks. Ultimately, embracing this paradoxical approach leads to better project outcomes.

13.2.7: Make Friends and Keep Them Friendly

A leader of a multibillion-dollar public sector project once told the authors he spent more than half his time **acting like a diplomat**, fostering the understanding and support of stakeholders who could significantly influence his project. Why? Because **it's risk management**. Stakeholder relationships can be as critical to project success as technical proficiency.

image by caiaimage/sam edwards from istock

Consider an IT project to build a company-wide data lake. From the

CTO to the heads of various business units and the security team, each stakeholder has a vested interest in the project. The CTO is focused on innovation and delivery, business unit leaders are concerned about how it will impact their workflows, and the security team worries about data governance and compliance. If one of these groups becomes disengaged or adversarial, it could stall the project at a critical juncture. By taking the time to understand and address each stakeholder's concerns early, you **create allies** who are **more likely to help resolve issues** when they inevitably arise.

Take another example: integrating a legacy system with a new ERP platform. The legacy system might be critical to a department that feels threatened by change. Building relationships with the department head, listening to their concerns, and involving them in key decision-making processes will ensure their cooperation when the integration hits a snag, whether related to data compatibility or performance issues.

When a project hits a roadblock—a budget overrun, a technical issue, or a timeline delay—the strength of your relationships with stakeholders can determine whether you'll get the support you need to solve the problem or face resistance. And **when something goes wrong, it's too late to start** building those relationships. If the data lake suffers performance issues or the ERP integration causes workflow disruptions, you'll want those stakeholders in your corner, ready to collaborate instead of criticize.

The lesson is clear: **build your bridges before you need them**. Invest time cultivating goodwill and understanding with those who can impact your project. Their support could be the key to turning a potential project failure into a success.

13.2.8: Watch Your Downside

It's often said that opportunity is as important as risk. However, this is a misconception. The truth is, **risk can significantly disrupt your project**, and **no potential upside can compensate** for catastrophic failure. This is particularly crucial in the face of fat-tailed risks—rare but severe issues that can have a disproportionate impact. Instead of relying on forecasting risk probabilities, your focus should be on risk mitigation. Failure to manage these risks could lead to severe consequences for your project.

image by glenn_hewitt from istock

For example, consider a project to migrate a company's entire infrastructure to the cloud. On paper, the opportunities are massive—scalability, cost savings, and improved performance. However, the risks, such as data loss during migration, misconfigurations that lead to security vulnerabilities, or unexpected downtime, could spell disaster. It doesn't matter how great the benefits are if a critical outage compromises your customer's trust or results in costly compliance violations. A wise project leader in this situation would focus first on ensuring robust data backup strategies, rigorous testing, and failover systems before getting excited about the potential upside of cloud performance.

Another example is implementing a microservices architecture. While microservices offer the promise of flexibility and scalability, it's crucial to remember the downside risks. These include service failure due to mismanagement of dependencies, network latency issues, or even cascading failures across services. The key here is not to get carried away by the agility and speed of deploying new features, but to first ensure that the system is resilient. This means building robust monitoring, automating redundancy, and preparing for the worst-case scenarios, such as service outages or database failures.

Flyvbjerg and Gardner use the example of a rider in the grueling three-week Tour de France bicycle race who explained that participating is

not about winning but not losing, day after day for twenty-one days. Only after that can you consider winning. Successful IT project leaders think the same way; they focus on not losing—every day—by mitigating risks such as data breaches, performance bottlenecks, or regulatory non-compliance. Once you manage the risks, you can start thinking about optimizing for success.

In short, successful project leaders prioritize downside protection. They understand that the most significant opportunity doesn't matter if unchecked risks derail the entire effort. They focus on reducing risk and "not losing" each day while keeping an eye on the prize—the ultimate goal they are trying to achieve.

13.2.9: Know That Your Biggest Risk Is You

It's easy to think that projects fail because of external factors—scope changes, unforeseen technical issues, new regulations, or shifting management priorities. While these challenges certainly arise, they are not the root cause of most project failures. In reality, many projects fail because we fall **prey to our biases and blind spots**. Jim Lasko's Great Chicago Fire Festival didn't fail because he couldn't predict the specific malfunction of the ignition system—it failed because he didn't take the *outside view*, which would have shown him how failure typically occurs in live events. He focused on his project's unique details and ignored the broader lessons from similar events.

image by martin barraud from istock

In IT, this **inside view** bias can manifest in many ways. A project manager might assume that a new system will integrate smoothly with existing infrastructure because they've seen their team solve integration challenges before, ignoring that the vast majority of similar projects encounter delays or failures during this phase. Or a software engineer might believe they can implement a cutting-edge technology stack because they're excited about its potential, disregarding the typical risks associated with introducing unproven tools.

Behavioral biases like overconfidence, optimism bias, and scope neglect can derail even the most well-intentioned project leaders. For instance, it's common to underestimate the time and resources needed to complete a complex software build, leading to missed deadlines and blown budgets. You might believe that your team is different or that your project is unique, but this thinking ignores the wealth of data from countless failed projects that followed the same pattern.

In software development, for example, how often have ambitious timelines slipped because someone believed the team could handle it? Or how often has scope creep been justified by the belief that "just one more feature" won't cause significant delays? These are classic inside-view mistakes. To avoid them, take the outside view—consider your project as part of a larger class of similar efforts. How do these projects typically

fail? What risks do they usually face? Use historical data, reference-class forecasting, and learn from the failures of others. This learning process can empower you to make better decisions and improve your project outcomes.

A practical example: if you're leading a project to implement a new enterprise resource planning (ERP) system, resist the temptation to believe that your team will avoid the common pitfalls because they're talented or because this project feels different. Look at how ERP projects typically fail—data migration issues, resistance to change, misalignment with business processes—and plan accordingly. Acknowledge that these risks apply to your project, too, and focus on mitigating them from the start.

By acknowledging that your biggest risk is you—your biases, assumptions, and overconfidence—you can make better decisions. You can mitigate risks by looking beyond the specifics of your project and learning from the broader class of similar projects. This shift in perspective not only helps you avoid common pitfalls but also opens up a world of potential success. It can make the difference between a successful project and a failed one.

13.2.10: Say No and Walk Away

Staying focused is critical for getting projects done, especially in IT projects, where **complexity can spiral quickly**. The **ability to say no** is essential for maintaining that focus. At the outset of any project, you must assess whether you have the people, the budget, and the contingencies needed to succeed. If the answer is no, you must have the discipline to walk away, or at the very least, reconsider or delay the project until the conditions are right.

image by deepak sethi from istock

Saying no early on protects you from a variety of pitfalls. For example, if a proposed system integration lacks an adequate budget or experienced engineers, pushing ahead will likely result in delays, technical debt, or outright failure. Declining or rescoping the project is better than proceeding with insufficient resources. Similarly, during the project, continuously ask: *Does this action directly contribute to achieving the goal?* If it doesn't, skip it. This discussion means saying no to **monuments**—features or systems that look impressive but add no real value—or to **untested technologies** that might introduce unnecessary risk without a clear payoff.

One common temptation in software development is **adopting the latest and greatest technology**, even when it's **not mature or well-suited** to the project. This temptation can lead to unnecessary complexity and time spent troubleshooting rather than building. If the technology doesn't directly support your core objectives, say no and stick with proven solutions. Likewise, you should say no to **scope creep**—the gradual expansion of a project beyond its original goals. It's easy to get sidetracked by requests for new features or integrations, but every additional element increases complexity, introduces new risks, and stretches your resources.

Saying no can be especially difficult in organizations with a **bias for action**, where the pressure is on to deliver and move fast constantly. But

success often comes from what you don't do. Steve Jobs once said, "I'm as proud of the things we haven't done as the things we have done." Apple's ability to stay focused on just a few products allowed them to perfect those products and achieve massive success. In the same way, IT projects succeed when teams focus on doing fewer things but doing them well.

In other cases, saying no isn't just about scope or technology—it's about **risk**. For instance, if a project exposes your company to potential legal issues (such as data privacy concerns or regulatory non-compliance), saying no early on is critical. Even if the project seems valuable, the risk of lawsuits or regulatory fines may far outweigh any potential benefits. Recognize when it's time to walk away.

Ultimately, **saying no is about discipline**. It keeps the team focused on the primary goal, preserves resources, and ensures that the project stays on track. While saying no can feel difficult at the moment, especially when everyone is pushing to do more, it's often the key to long-term success. By cutting out distractions, avoiding unnecessary risks, and focusing on what truly matters, your project and your organization will be better positioned to succeed.

13.2.11: Build Climate Mitigation Into Your Project

No task is more urgent today than mitigating the climate crisis. **The time to act is now**, not only for the common good but for your organization, yourself, and your family. In the realm of IT projects, climate action should be a core consideration for any project you undertake.

image by francesco scatena from istock

For IT and software engineers, this might mean focusing on **energy efficiency** in data centers, applications, and infrastructure. For example, consider not just performance and cost but also the environmental impact when designing a cloud architecture. Choose cloud providers that prioritize renewable energy in their data centers. Many leading cloud platforms, such as AWS, Google Cloud, and Microsoft Azure, have committed to sustainability goals and offer carbon-neutral options. By selecting these, you reduce the carbon footprint of your project while maintaining its technical excellence.

Another practical step is **electrifying everything** and ensuring that your IT infrastructure's electricity comes from renewable sources. This step could involve shifting to more energy-efficient hardware and systems that consume less power. In software, you can contribute by optimizing code and reducing unnecessary computation, which cuts energy usage. If your organization has its data centers, you can advocate for a shift to renewable energy sources, such as solar or wind, to power them.

A concrete example might involve designing a green software project from the ground up. This involves adopting principles like sustainable coding, where algorithms are optimized for lower energy consumption, and using tools to monitor your application's energy impact. Additionally, consider how AI and machine learning models are deployed—these technologies

are notoriously resource-intensive. Implement energy-efficient practices for training and running models, such as using specialized hardware designed for energy efficiency or scheduling resource-heavy processes when renewable energy supply is highest.

We already have the technology and knowledge to implement these strategies—what's left is scaling them up across thousands of projects, large and small. This is where you, as an IT leader or engineer, play a vital and empowering role. Every system you design, every platform you build, and every line of code you write can contribute to a more sustainable future. Whether it's through reducing waste, minimizing power consumption, or supporting renewable energy, the opportunities for climate mitigation in IT are vast.

Climate mitigation should no longer be seen as an optional "nice to have" in project planning. It's a fundamental part of responsible engineering and project management. Integrating sustainability into your IT initiatives helps accelerate the global transition to a low-carbon future. Following the principles laid out in this book, you can ensure your projects succeed on a technical level and contribute to the greater good.

13.3: To Probe Further

- How Big Things Get Done[3], by Bent Flyvbjerg and Dan Gardner, 2023
- The Mythical Man-Month[4], by Fred Brooks, 1975
- Sketching User Experiences: Getting the Design Right and the Right Design[5], by Bill Buxton, 2007

[3] https://www.amazon.ca/How-Big-Things-Get-Done/dp/077109843X
[4] https://en.wikipedia.org/wiki/The_Mythical_Man-Month
[5] https://www.amazon.com/Sketching-User-Experiences-Getting-Design/dp/0123740371

13.4: Questions to Consider

- *What strategies can be employed to avoid strategic misrepresentation and ensure that initial project forecasts are honest and achievable?*
- *In what ways can IT leaders apply the lessons learned from other industries (e.g., construction, entertainment) to improve the success rates of large IT projects?*
- *How can organizations identify and address "sunk cost" and "commitment fallacies" in their IT transformation projects?*
- *What are the qualities of a "master builder" in IT transformations, and how can organizations ensure they have the right leadership in place?*
- *How can IT teams stay focused on the core "why" of the transformation and avoid being sidetracked by unnecessary features or scope creep?*
- *How can organizations apply the "build with Lego approach" in IT projects to create scalable, modular systems?*
- *How can taking an outside view, or reference-class forecasting, help IT leaders better estimate timelines, costs, and risks in their projects?*
- *What strategies can IT leaders use to foster strong, collaborative relationships with stakeholders to support project success?*
- *What are the risks of failing to "watch your downside" in IT transformations, and how can organizations better prioritize risk mitigation?*
- *What role do cognitive biases, particularly overconfidence, play in IT transformation failures, and how can teams safeguard against them?*
- *How can climate mitigation be integrated into IT transformation projects, and why is this essential for the organization and society?*