

Introduction to Software Architecture

SOEN 343: Software Architecture and Design

Diego Elias Costa

Why do we need Software Architecture?



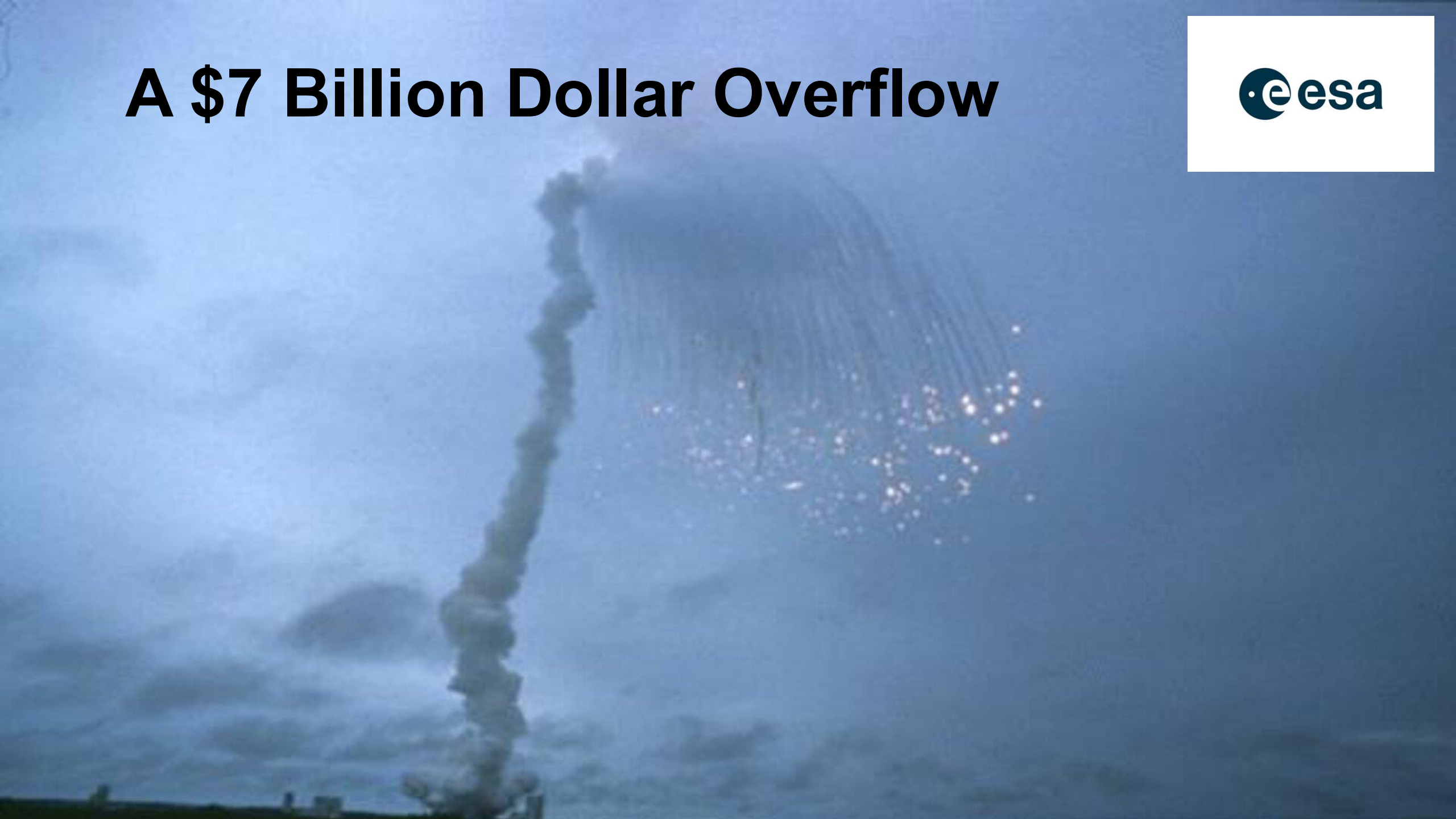
Knight

**HOW TO LOSE \$440M
IN 45 MINUTES**

How to lose \$440 million in 45 minutes

- Faulty software deployment reactivated old, unused code.
 - PowerPeg was a sanity check of the system, designed to trade at a deficit to check if the trading system was online.
 - During deployment, the PowerPeg code **was mistakenly reactivated** and triggered millions of orders.
- **Architectural oversight:**
 - The system lacked a proper modular design
 - Developers did not prune dead code
 - No safeguards or rollback mechanisms in design.
 - No monitoring
 - No CI/CD pipelines

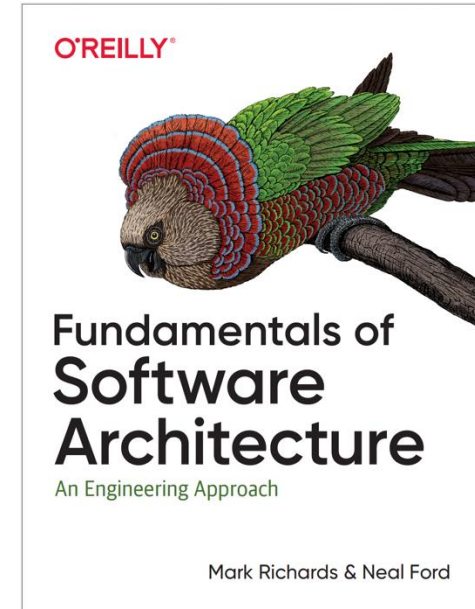
A \$7 Billion Dollar Overflow



Ariane 5 Failure

- The **Ariane 5 maiden flight** exploded 37 seconds after launch.
- Cause: a **software exception** in the navigation system.
 - A **64-bit floating-point number** (horizontal velocity) was converted into a **16-bit signed integer**, which overflowed.
 - The exception crashed inertial systems → guidance failed → rocket veered off course and self-destructed.
- **Architectural & Design oversight:**
 - Ariane 4 software was reused without understanding the Ariane 5 capabilities
 - Lack of Fault Tolerance in Architecture
 - Unnecessary Functionality Running
 - The code that failed was only needed before liftoff
 - Poor exception handling

What is Software Architecture?



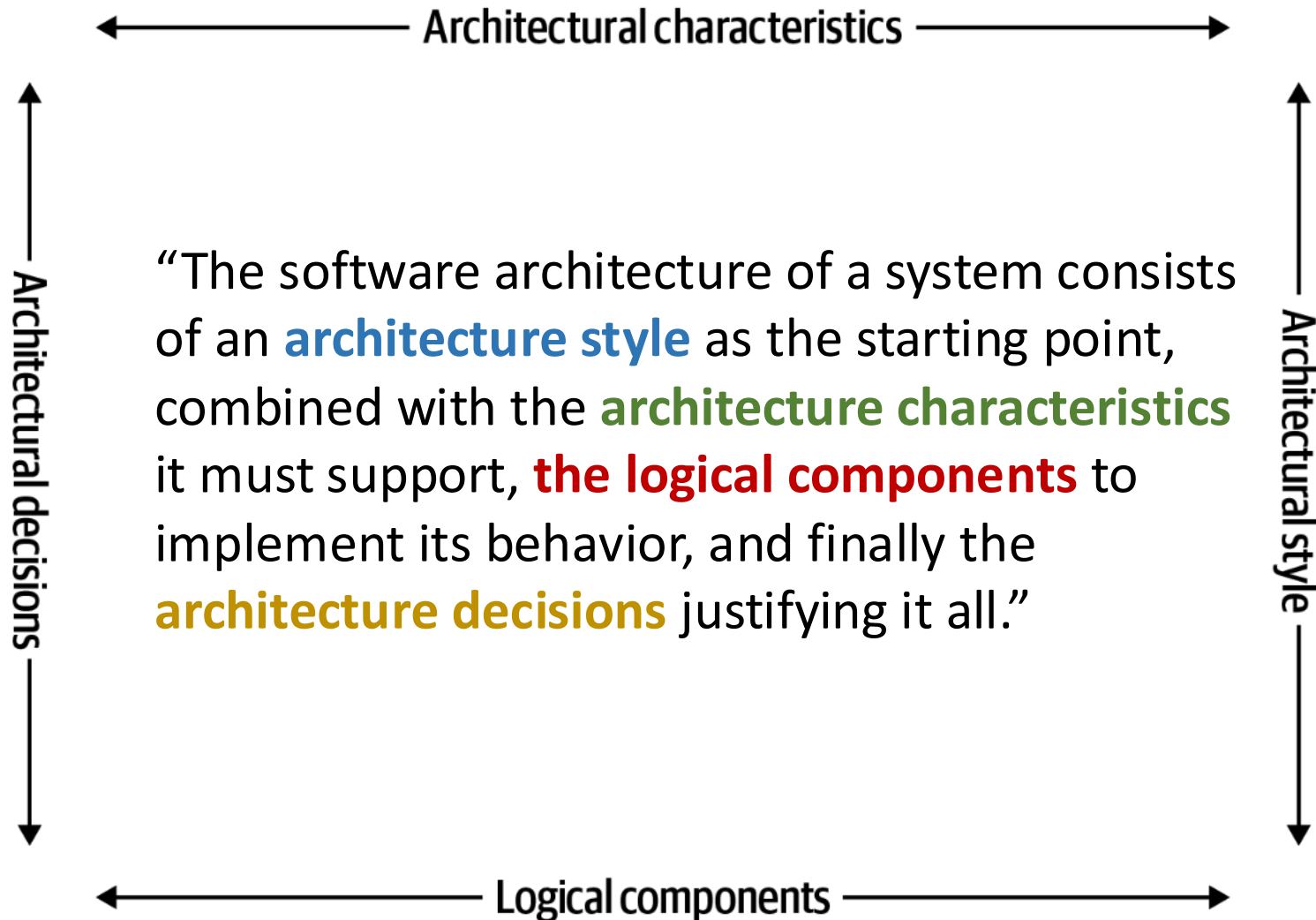
Chapter 1. Introduction

Defining Software Architecture

“The fundamental organization of a system embodied in its **components**, their **relationships** to each other and to the environment, and the **principles** guiding its design and evolution.”

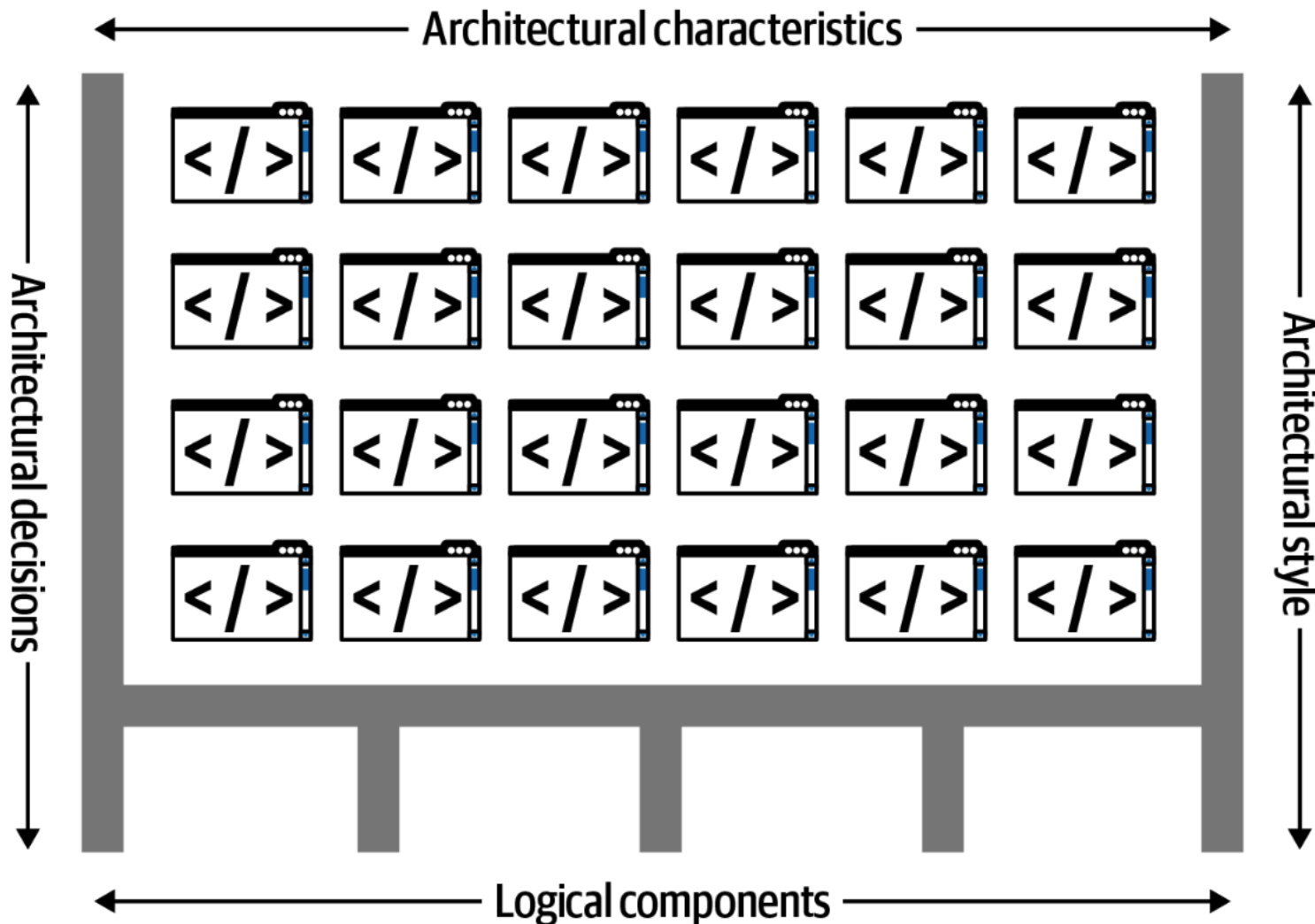
IEEE 1471

Dimensions of Software Architecture

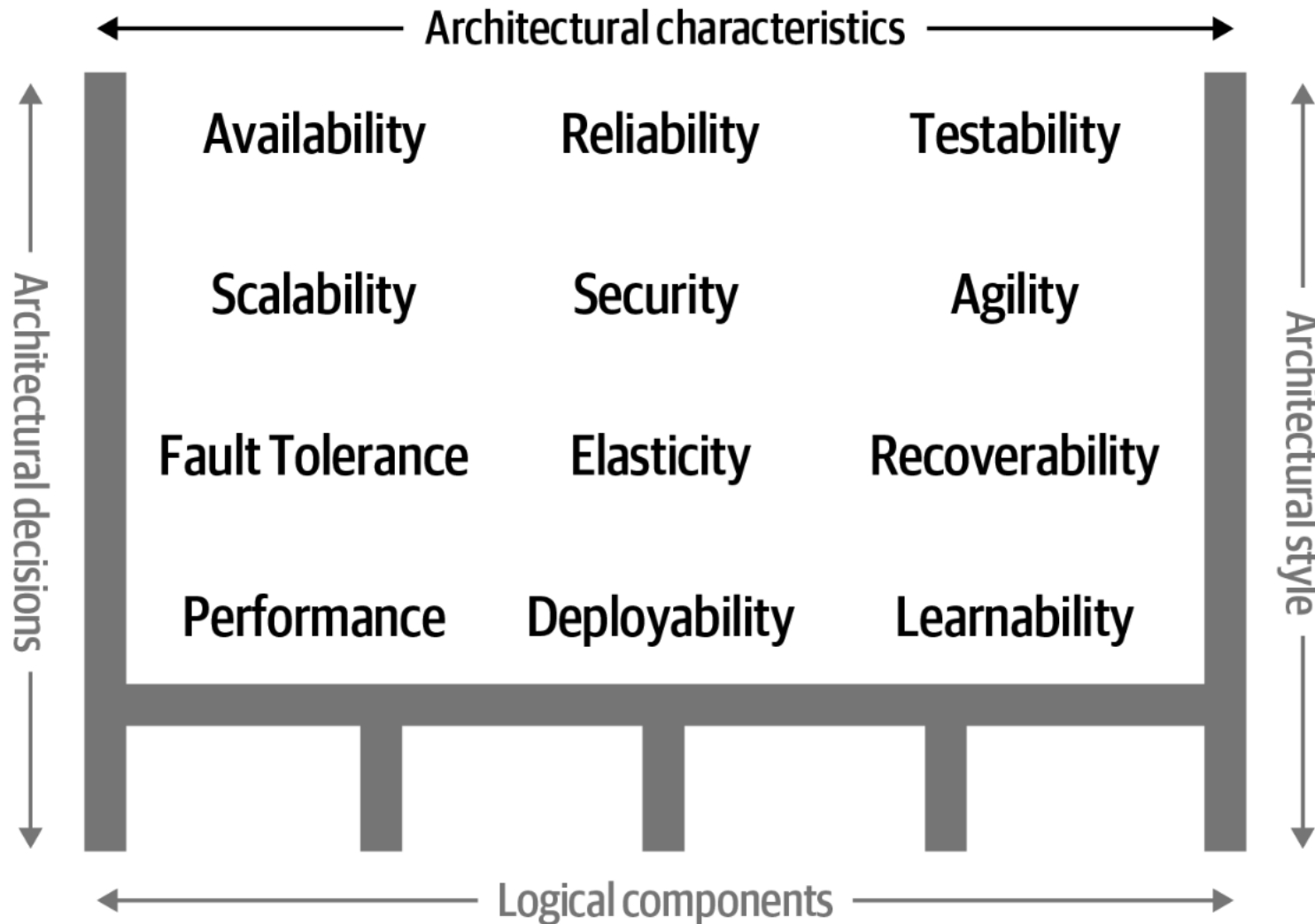


Dimensions of Software Architecture

(continued)

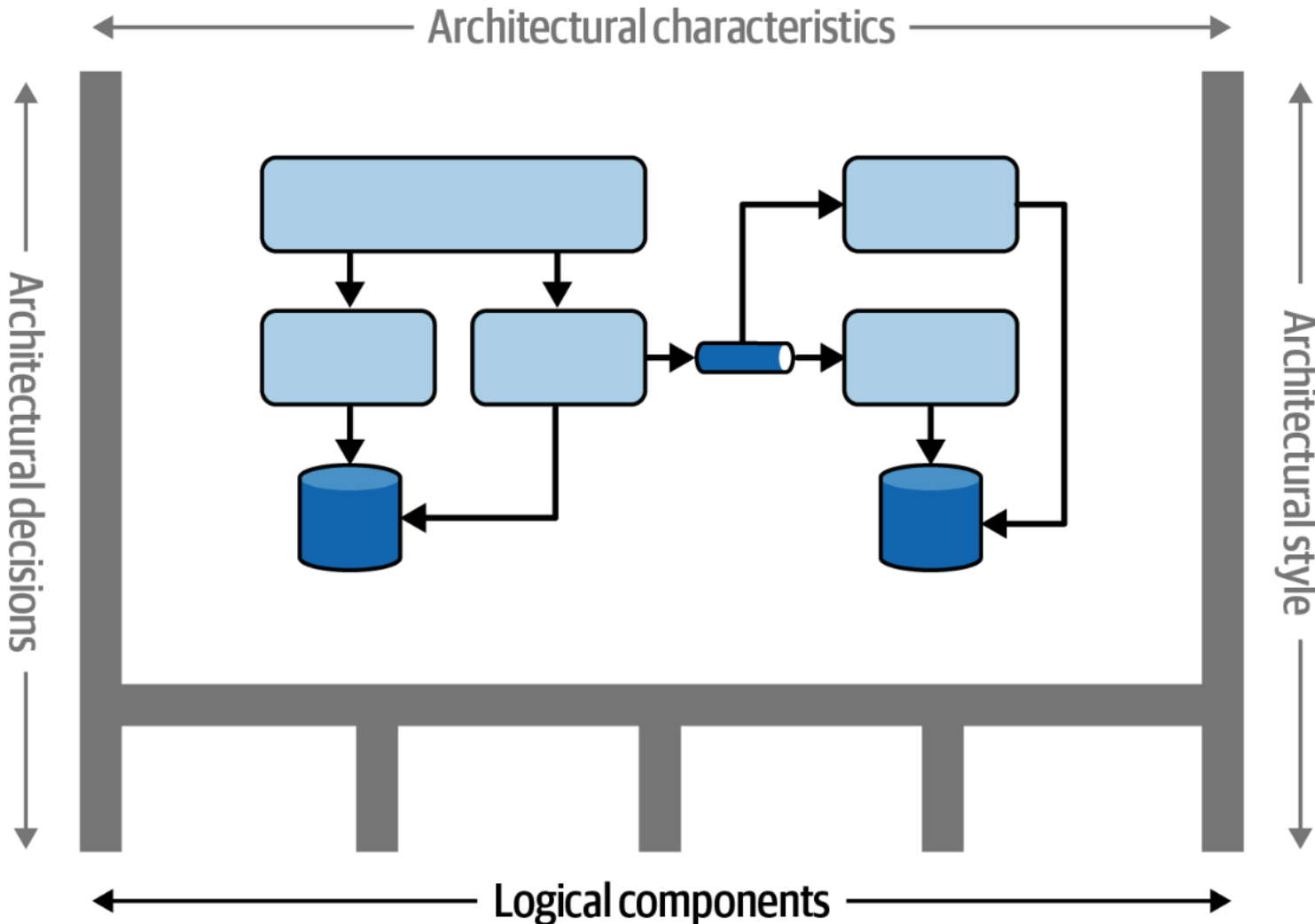


Architecture Characteristics



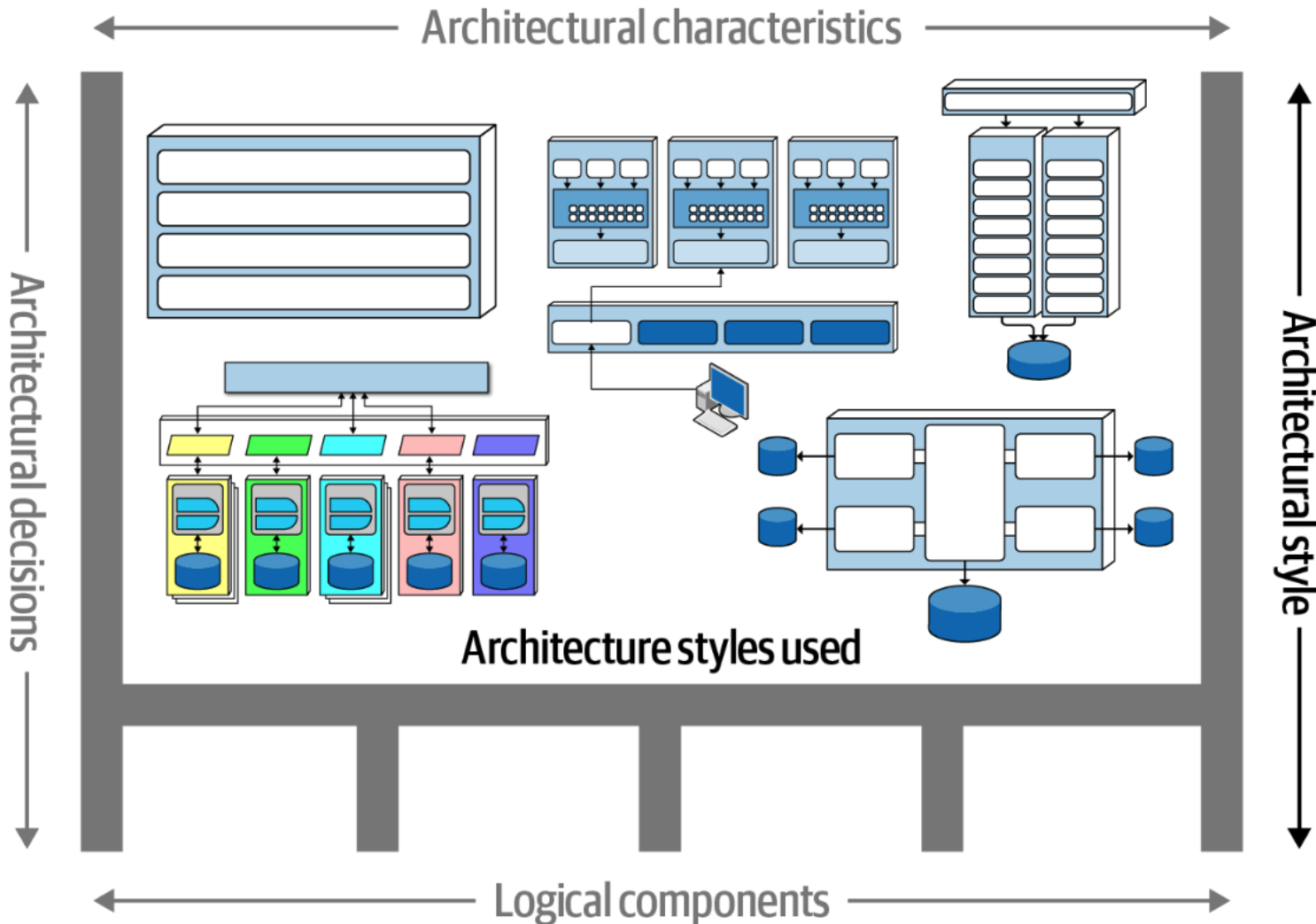
Architectural Characteristics define the **capabilities** of a system

Logical Components



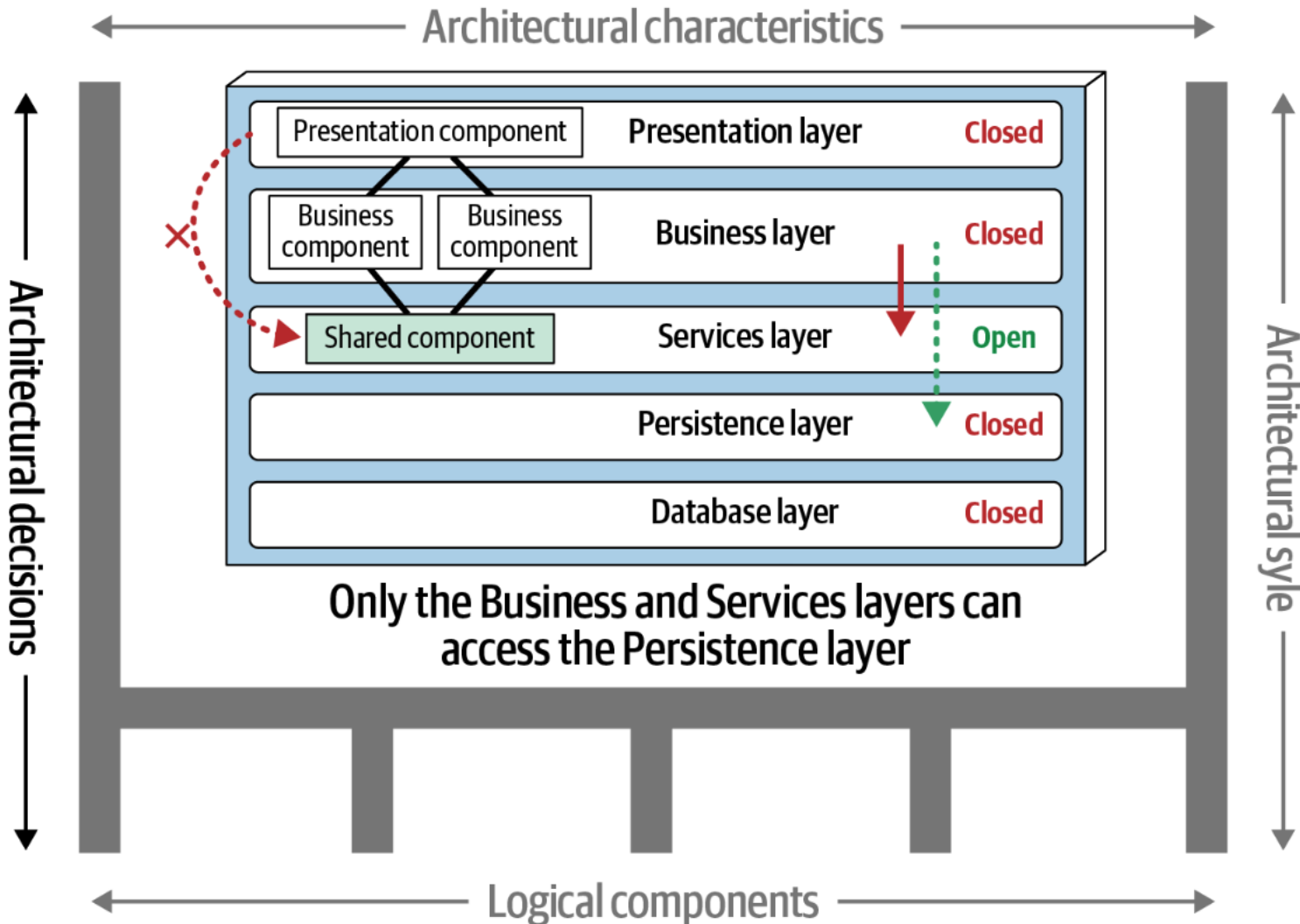
Logical components define the system's **behavior**.

Architecture Styles



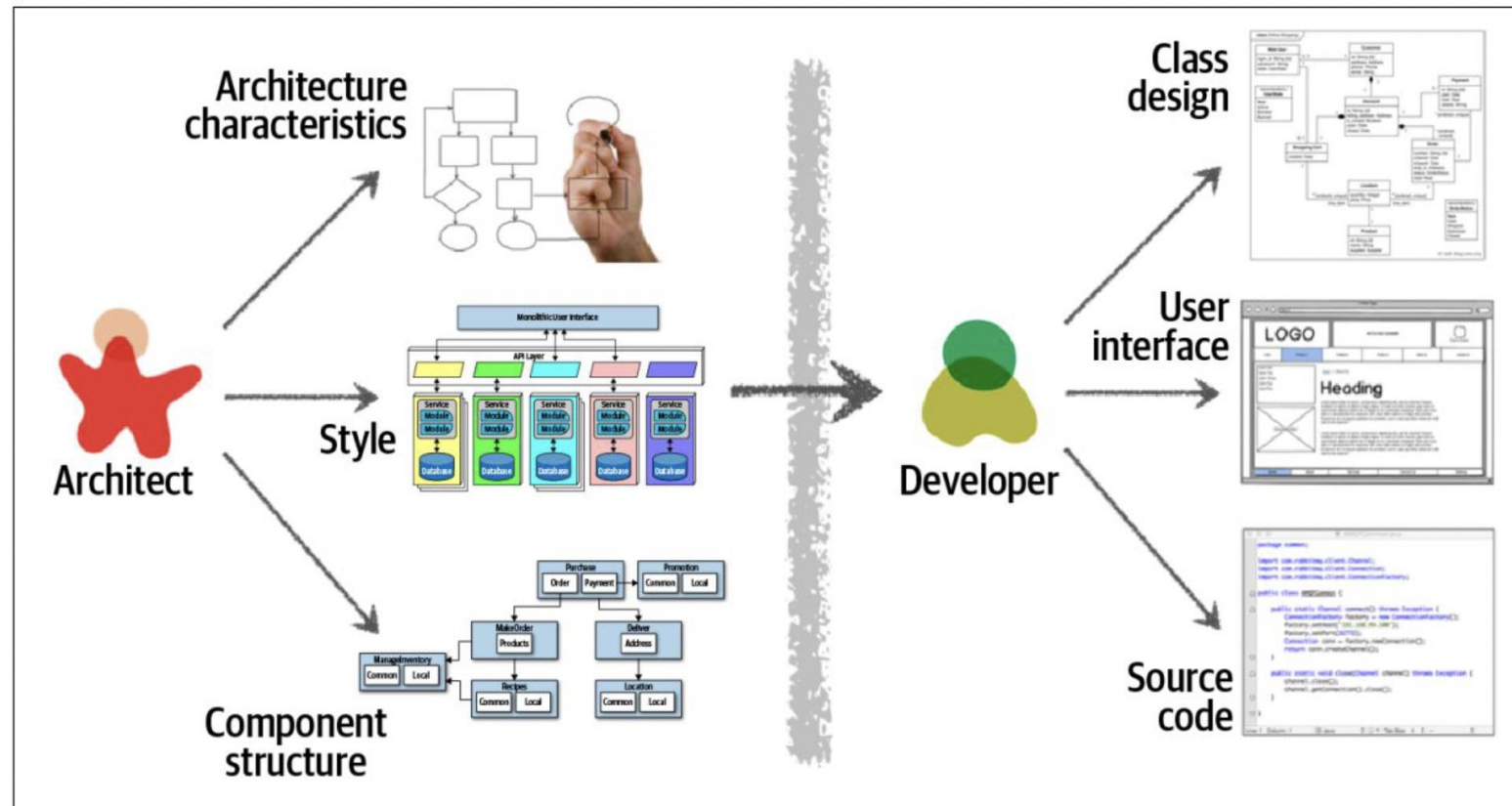
The architecture style defines the **implementation path** for a given set of requirements.

Architecture Decisions

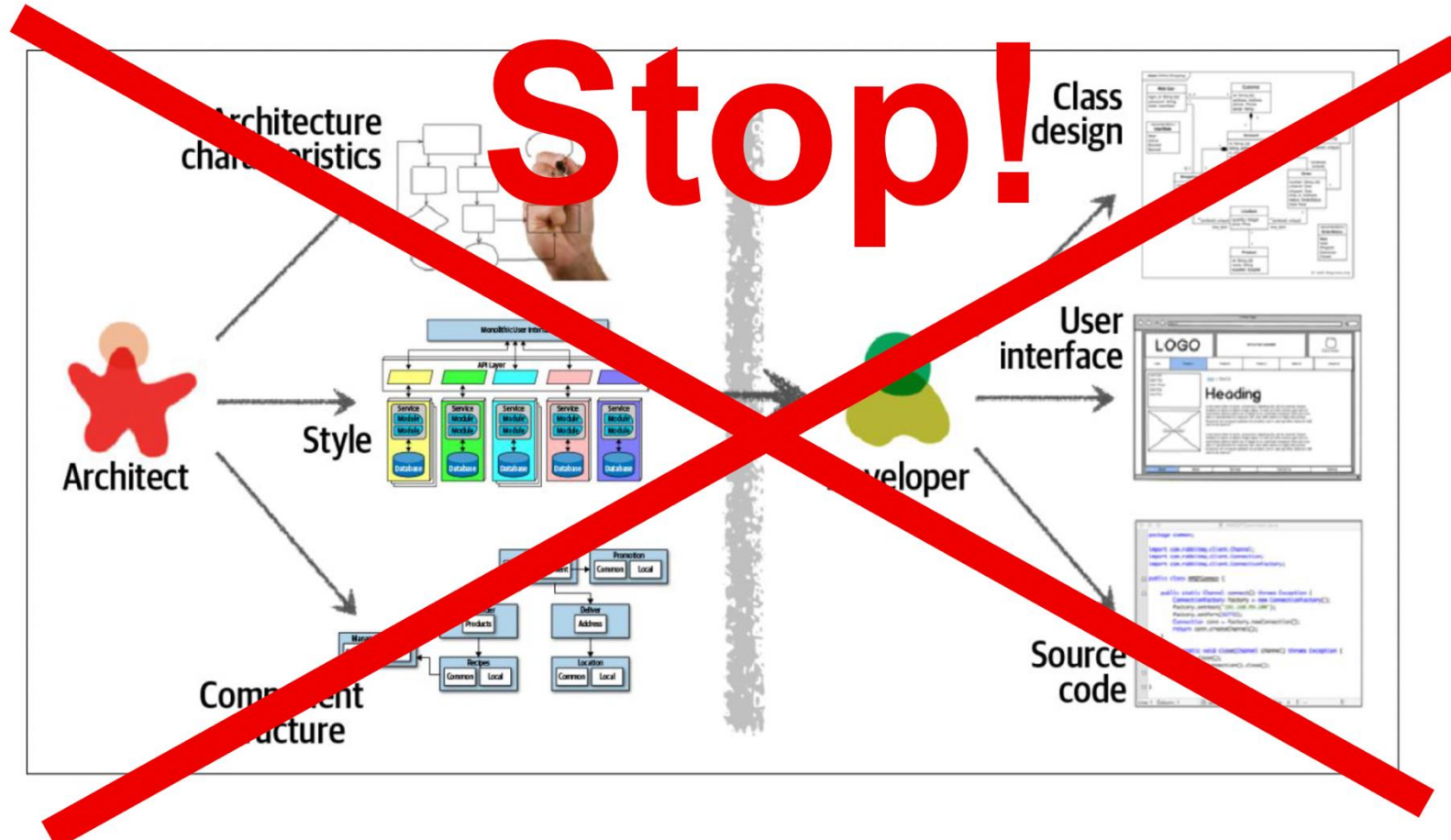


The architecture decision describes **the rules** for how a system should be constructed.

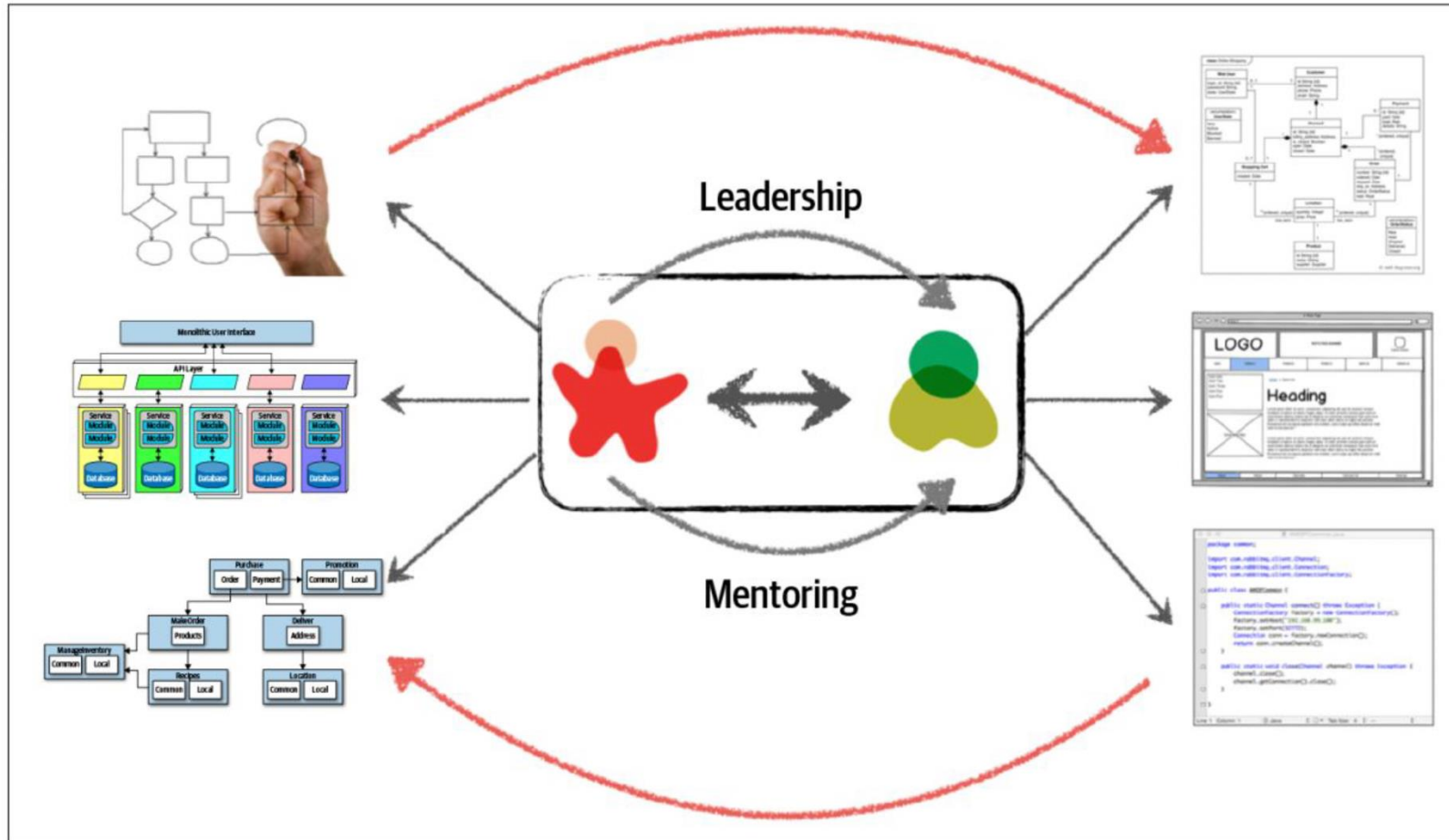
Traditional view of: Architecture vs Design



Traditional view of: Architecture vs Design



Making architecture work through collaboration



First Law of Software Architecture

“Everything in software architecture is a **trade-off**.”

“If you think you’ve discovered something that isn’t a trade-off, more likely you just haven’t identified the trade-off...yet.” Corollary 1

“You can’t just do trade-off analysis once and be done with it”. Corollary 2

Second Law of Software Architecture

“Why is more important than how.”

Third Law of Software Architecture

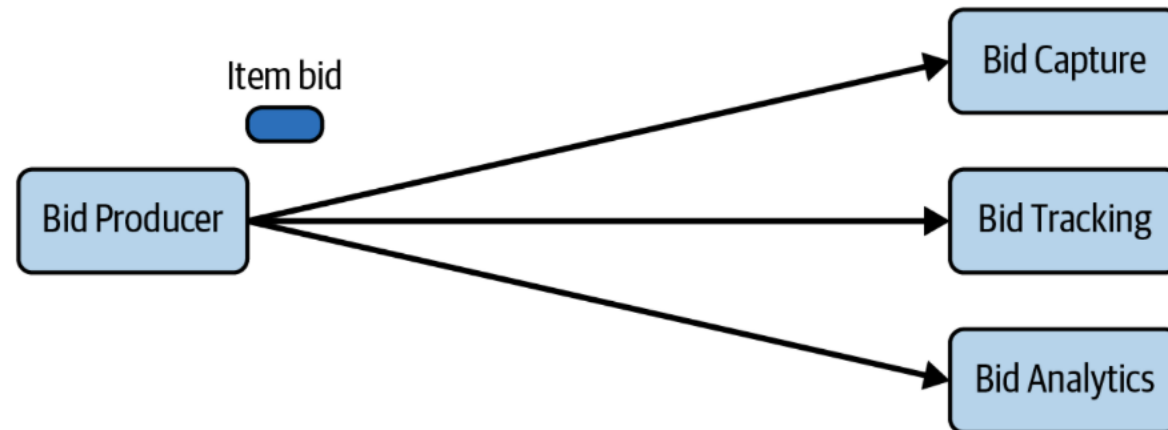
“Most architecture decisions aren’t binary but rather exist on a spectrum between extremes.”

Analyzing Trade-Offs

- Thinking like an architect is all about seeing trade-offs to determine the most appropriate solution.
- There are no right or wrong answers in architecture – only trade-offs.
- Thinking architecturally involves examining the benefits of a given solution, as well as analyzing the negatives or trade-offs associated with it.

The Auction System Example

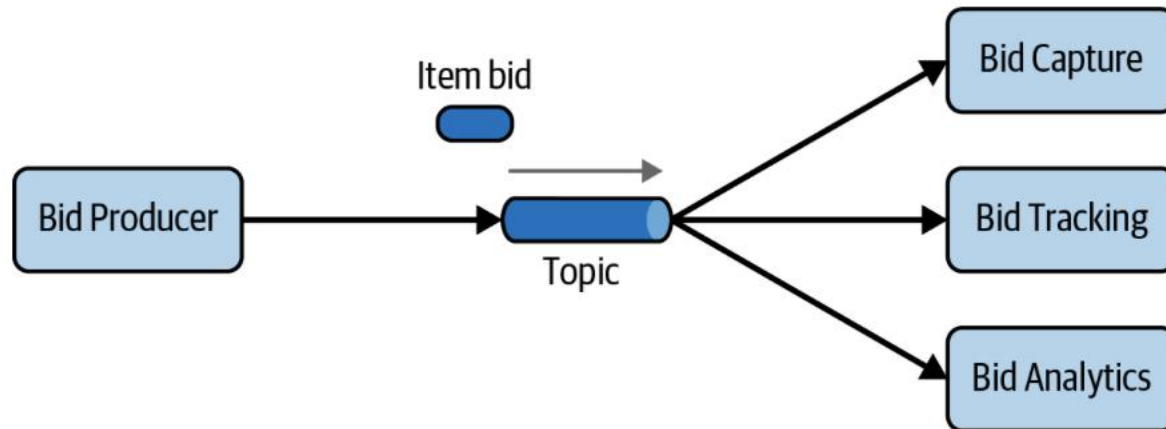
- An auction system where online bidders bid on items up for auction.
 - The Bid Producer generates a big from the bidder
 - Sends the bid amount to the services: Bid Capture Bid Tracking Bid Analytics



Two possible solutions

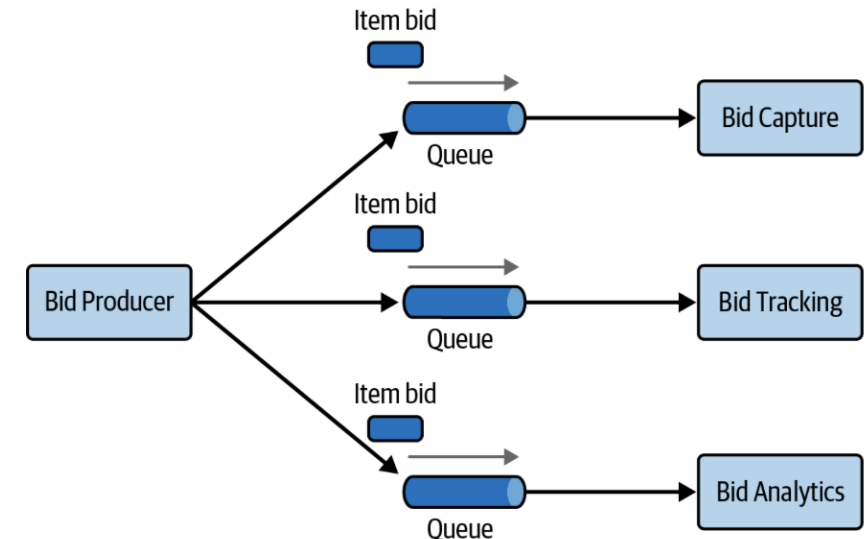
Using a topic (channel)

- Services subscribe to the channel and get notified when a new bid is submitted
- Publish-and-subscribe model



Using queues

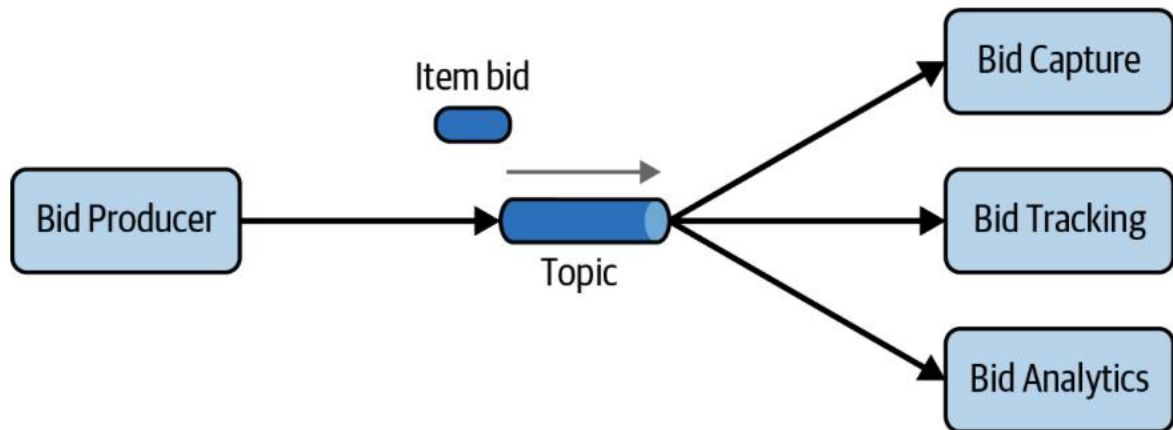
- Bid Producer sends the bid to individual queues to be consumed by their respective services
- Point-to-point communication



Trade-off Analysis

Using a topic (channel)

- Services subscribe to the channel and get notified when a new bid is submitted



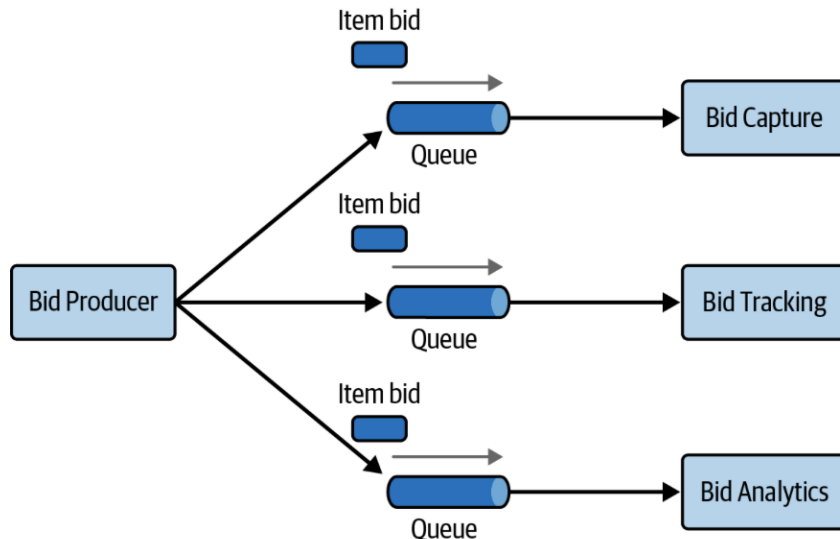
Advantages

- **Extensible**: Bid Producer requires just a single connection to the topic.
 - Easier to extend to new services (e.g., Bid History)
- **Decoupled**: Bid Producer does not know how the services will consume the item bid

Trade-off Analysis

Using queues

- Bid Producer sends the bid to individual queues to be consumed by their respective services
- Point-to-point communication



Advantages

- **More secure**: Bid Producer can control which service will receive the item bid.
- **Customized item bid**: Each service may receive only the information needed.
- **Allows for monitoring**: Queues can be monitored individually, and can be scaled separately.

Trade-off Analysis

- From the topic solution point-of-view

Table 2-1. Trade-offs for topics

Topic advantages	Topic disadvantages
Architectural extensibility	Data access and data security concerns
Service decoupling	No heterogeneous contracts
	Monitoring and programmatic scalability

Important Advice

“Don’t try to find the **best design** in software architecture. Instead, strive for the **least-worst combination** of **trade-offs**.”

Expectations of an Architect

- Make architecture decisions
- Continually analyze the architecture
- Keep current with the latest trends
- Ensure compliance with decisions
- Diverse exposure and experience
- Have business domain knowledge
- Possess interpersonal skills
- Understand and navigate politics

Software Developer salaries in Montreal, QC ⓘ

Experience

All years of experience

Base pay

\$64K - \$91K/yr

\$77K/yr Average base pay

Additional salary ⓘ

\$6K/yr Average

\$3K - \$11K/yr Range

Software Architect salaries in Montreal, QC ⓘ

Experience

All years of experience

Base pay

\$95K - \$132K/yr

\$112K/yr Average base pay

Additional salary ⓘ

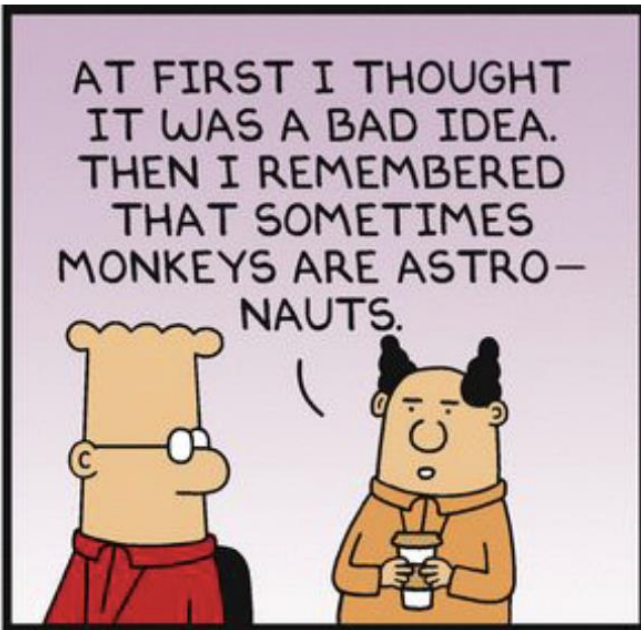
\$10K/yr Average

\$5K - \$19K/yr Range

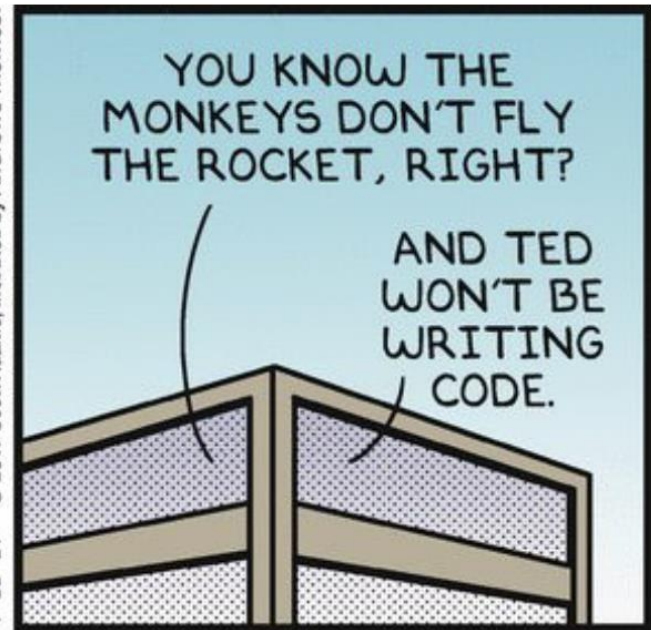
Source: Glassdoor



Dilbert.com @ScottAdamsSays



7-18-17 © 2017 Scott Adams, Inc./Dist. by Andrews McMeel



Source: <http://dilbert.com/strip/2017-07-18>

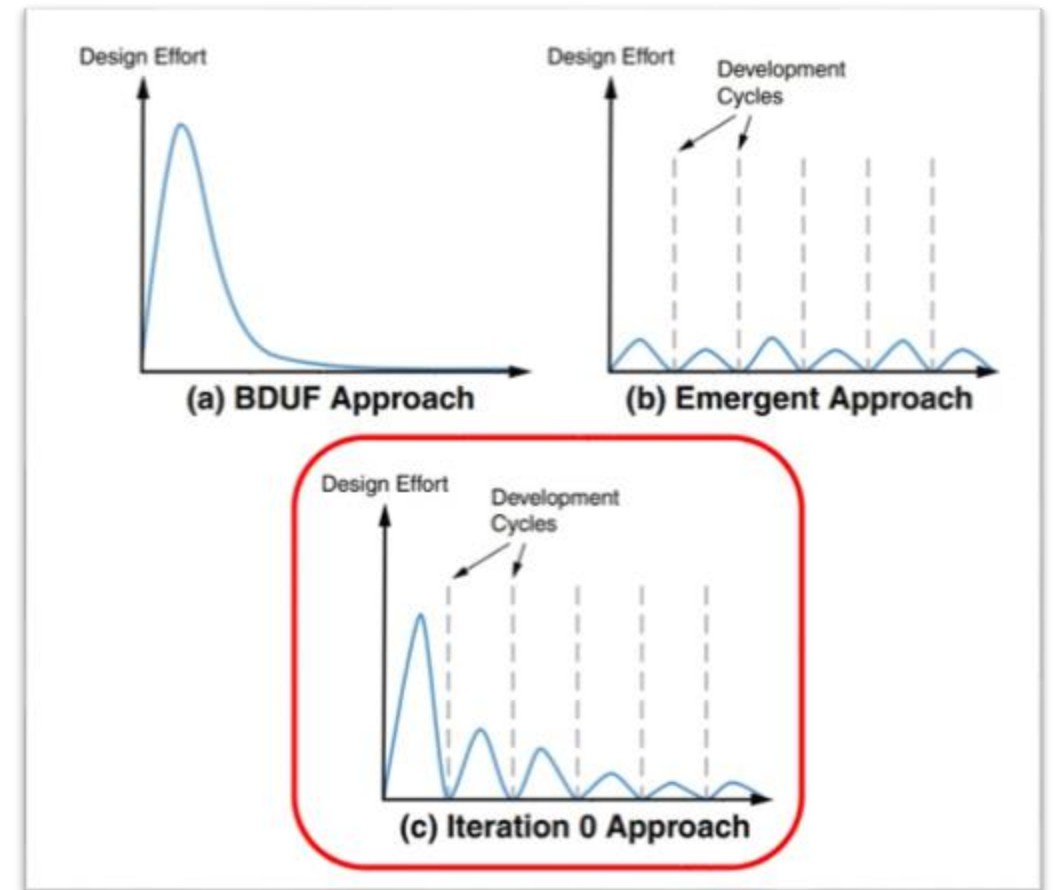
Make Architecture Decisions

“An architect is expected to define the architecture decisions and design principles used to guide technology decisions within the team, the department, or across the enterprise.”

- E.g., Use a reactive-based framework rather than React.js, Angular, etc.

Continually Analyze the Architecture

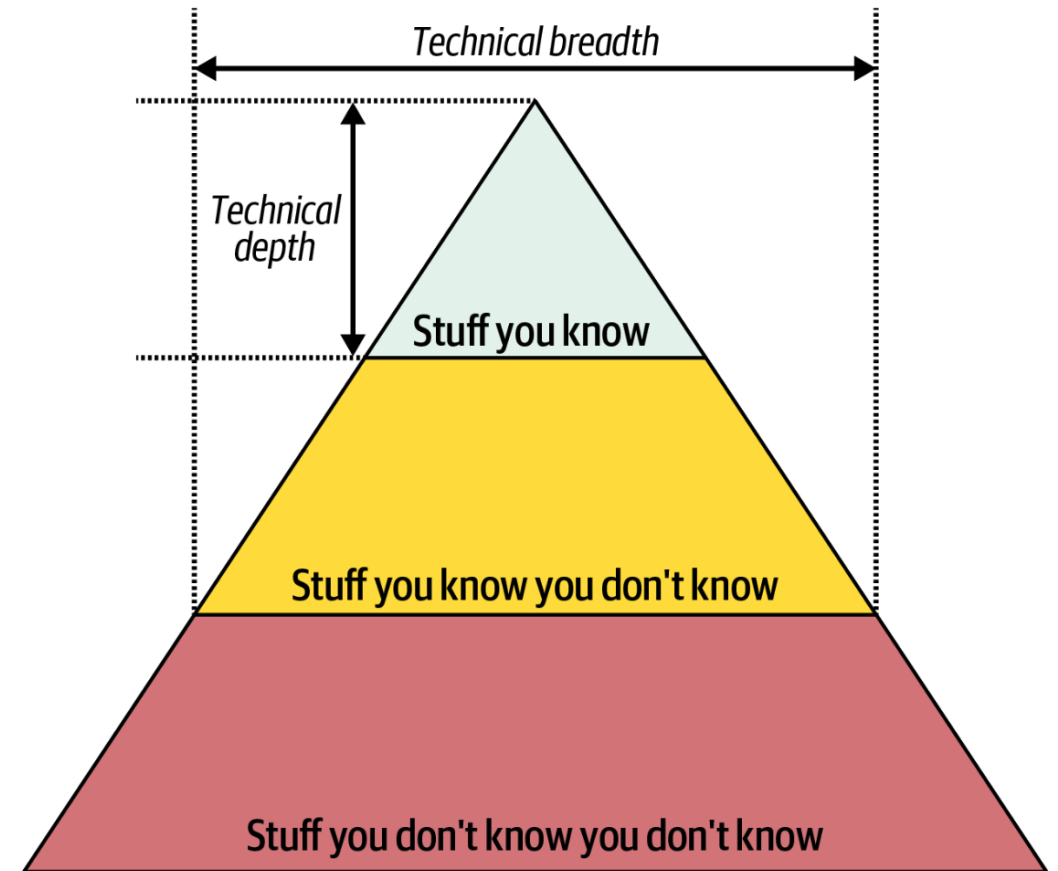
- How viable the architecture that was defined years ago is today given current changes in business and technology?
- Look for design decay (anti-patterns)
- Testing and releasing environments (DevOps)



Keep Current with Latest Trends

Architects need to maintain their breadth of knowledge

- Architectural decisions last long
- Following key trends to **anticipate changes** and make informed decisions



Participation activity 1: Life-long learning in SE

- This counts for the series of in-class participation activities during the semester, which adds up to 5% of the total grade of the course.
- Duration: 15 minutes.
- In groups of 3-5 people, discuss which are your favorite resources for staying informed (e.g., books, online courses, podcasts, conferences) about the latest trends in Computer Science/Software Engineering.
- **To get marks for this activity, post your reply individually to the corresponding forum in Moodle.** Include a brief overview of your recommended resources and explain how these have helped you stay current.

The 20-minute rule



- “The idea is to devote at least 20 minutes a day to learning something new or diving deeper into a specific topic.”
- Sources collected from class:

Ensure Compliance with Decisions

- “An architect is expected to ensure compliance with architecture decisions and design principles.”



Know the Business Domain

- “An architect is expected to have a **certain level** of business domain expertise.”
 - Strengthen your knowledge in business domain to effectively communicate with C-level executives and business users
 - Example: An architect at a large financial institution will demand knowledge about common financial terms: e.g., portfolio, overdraft, credit score, etc.

Possess Interpersonal Skills

- “An architect is expected to possess **exceptional** interpersonal skills, including teamwork, facilitation, and leadership.”

Class	Description [Rozanski Woods]
Acquirers	Oversee the procurement of the system
Assessors	Oversee conformance to standards and legal regulations
Communicators	Explain the system via documentation and training
Developers	Construct and deploy the system from its specifications
Maintainers	Manage the evolution of the system once operational
Suppliers	Provide hardware/software platform on which the system will run
Support staff	Assist users to make use of the running system
System administrators	Run the system once deployed
Testers	Check whether the system meets its specifications
Users	Define the system's functionality and use it once running

Architecture is most important when...

- The failure **risks** are high
- It is hard to design **acceptable solutions**
- Stakeholders demand **difficult quality attributes**
- You are working on a **new domain**
- Architecture can help **manage complexity**
- A flexible architecture can help a system **adapt to changes**

Software Design

Software Design

” Design is the process of defining the architecture, components, interfaces, and other characteristics of a system or component” and the result of [that] process.”

SWEBOK V3

- **Software detailed design:** specifies each component in sufficient detail to facilitate its construction.

SWEBOK V3

Why do we need Software Design?

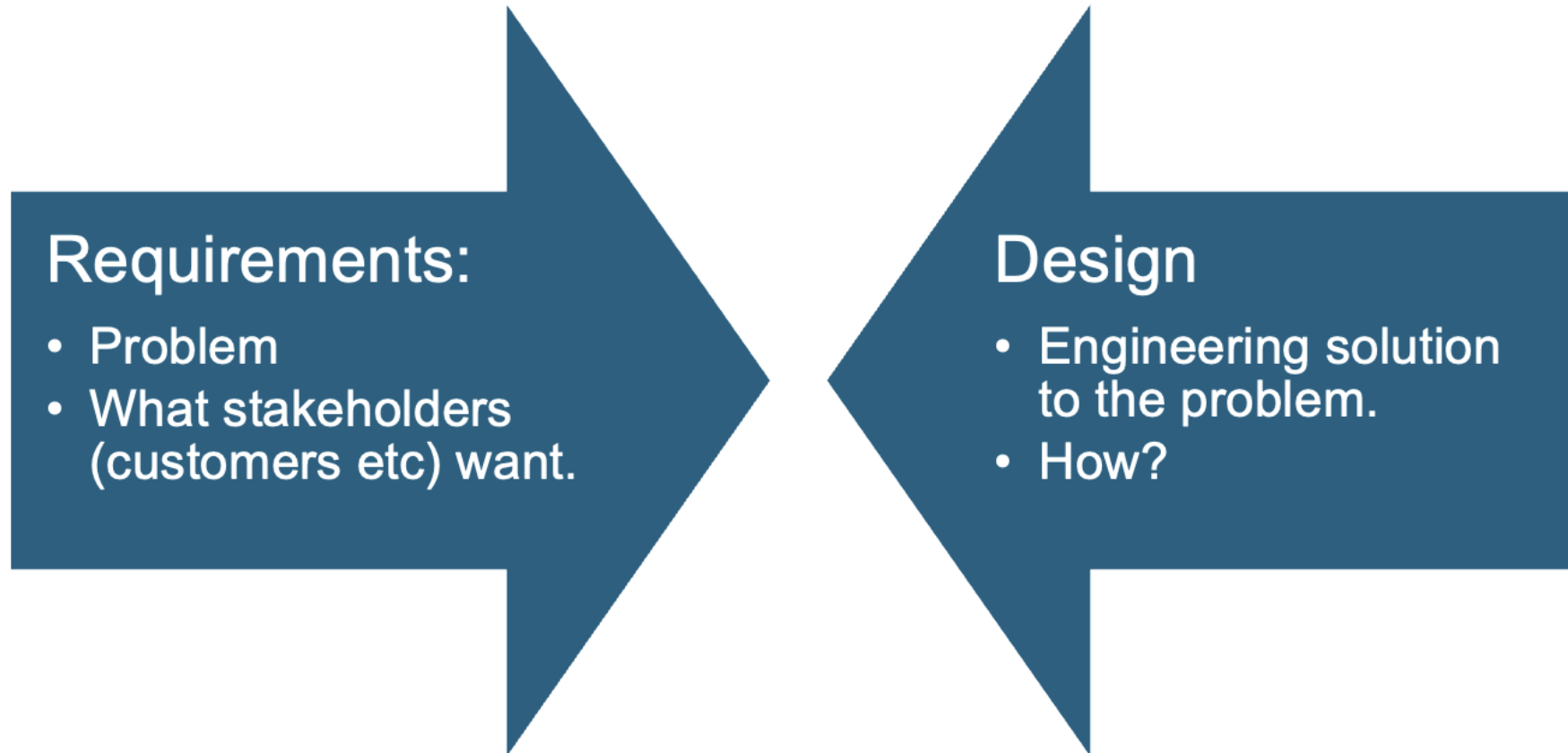
Because we know how to build, but requirements **change over time**:

- Helps manage complexity
 - Break large problems into smaller modules
- Improves Communication
 - Share blueprint of solutions (design patterns)
- Enables reuse and extensibility
 - Promoting modular and adaptable components
- Reduces Cost and Risk
 - Plan the design to prevent costly late-stage rework

Software Architecture vs Software Design

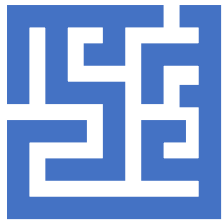
	Software Architecture	Software Design
Scope	High-level structure of the system	Detailed design of modules/classes
Focus	How the system is organized into components and how they interact	How individual components are implemented
Abstraction	“Big picture” – system decomposition, communication, deployment	“Zoomed in” – data structures, algorithms, design patterns
Stakeholders	Architects, project managers, clients	Developers, testers

Design vs Requirements



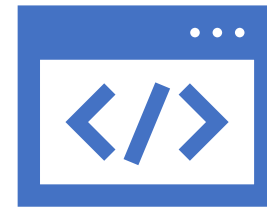
Building the right system vs. building the system right

Object-Oriented Analysis and Design



During object-oriented analysis there is an emphasis on finding and describing the objects—or concepts—in the problem domain.

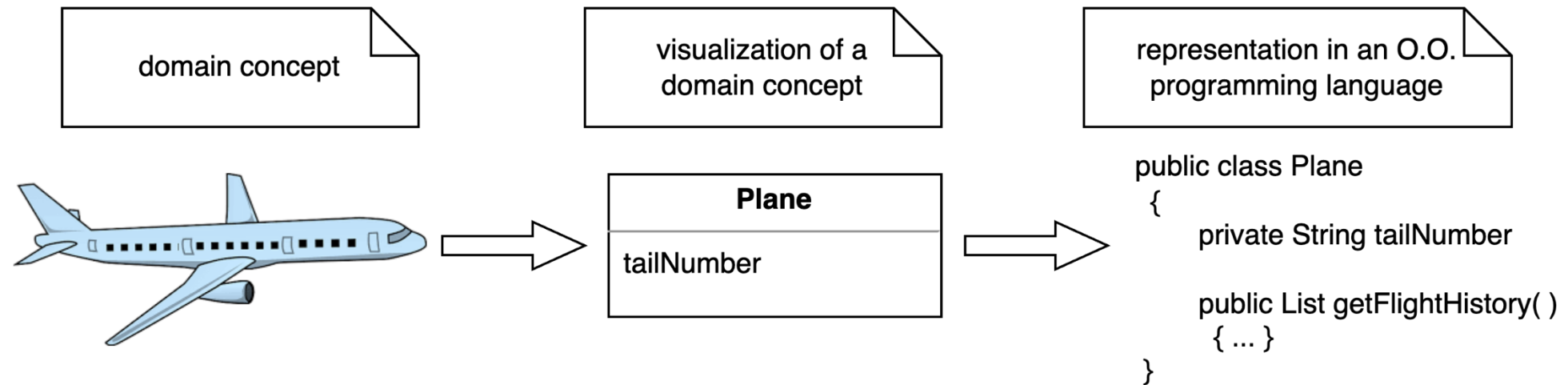
For example, in the case of a flight information system, some of the concepts include Plane, Flight, and Pilot.




During object-oriented design (or simply, object design) there is an emphasis on defining software objects and how they collaborate to fulfill the requirements.

For example, a Plane software object may have a tailNumber attribute and a getFlightHistory method

From analysis to implementation: an example



Ok, I get it. So how do we start architecting?



To Be Continued

