

# Architectural Characteristics

COMP2511, CSE, UNSW

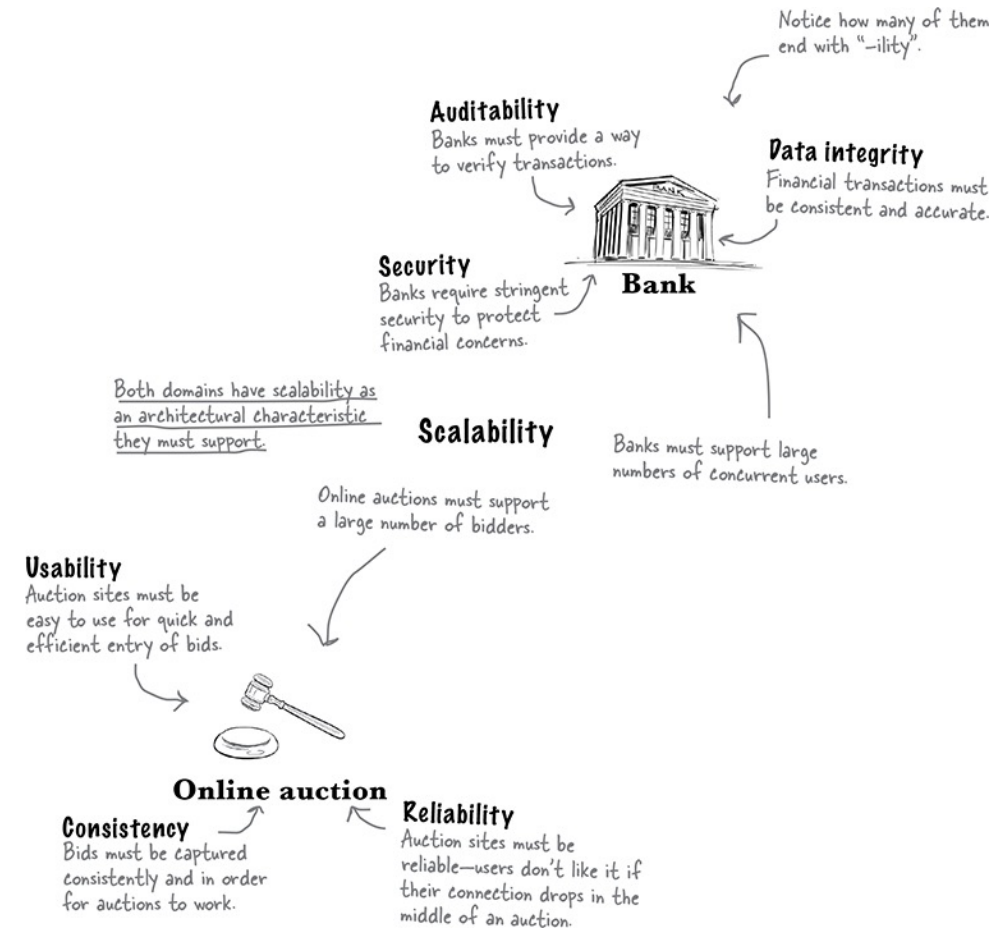


**UNSW**  
SYDNEY

These lecture slides are from the book “*Head First Software Architecture*”,  
by Raju Gandhi, Mark Richards, Neal Ford, O'Reilly Media, Inc., March 2024

# What Are Architectural Characteristics?

- ❖ Architectural Characteristics define **fundamental qualities** software architecture must support.
- ❖ They are often **not explicitly defined**
- ❖ They **influence structure**, infrastructure, and quality of the system
- ❖ Architectural characteristics **guide decisions** like architectural style, deployment, and scalability.



# Some of the Popular Architectural Characteristics

- |  |   |
|--|---|
| <ul style="list-style-type: none"><li>○ <b>Scalability</b>: Handles growth in traffic or data size</li><li>○ <b>Availability</b>: Ensures system uptime</li><li>○ <b>Maintainability</b>: Easy to update, fix, or extend</li><li>○ <b>Security</b>: Prevents unauthorized access</li></ul> | <ul style="list-style-type: none"><li>○ <b>Elasticity</b>: Automatically adjusts resources based on load</li><li>○ <b>Deployability</b>: Enables safe, frequent updates</li><li>○ <b>Responsiveness</b>: Provides quick feedback to users</li></ul> |
|--|---|

- ❖ There is **no fixed** or exhaustive **list** — they evolve with time and context
- ❖ Many required characteristics are **not explicitly stated** in the requirements
- ❖ Proper **domain** and **contextual** understanding is essential **to identify** them
- ❖ They are **essential to align architecture** with real-world needs, for successful development, deployment, and future system resilience

# Case Study - Lafter (Sillycon Symposia)

- ❖ A fictional tech-comedy conference platform to illustrate architectural thinking.
- ❖ **Use Case:** Participants post jokes/puns, react with "*HaHa*" or "*Giggle*," and engage with speakers.
- ❖ **Functional Needs:**
  - User registration
  - Posting content (jokes/puns)
  - Reaction buttons and speaker tools
  - Language support (international audience)
- ❖ **Architectural Constraints:**
  - **Scalability:** Must handle traffic bursts during peak conference hours.
  - **Availability:** Downtime would impact live sessions and user interaction.
  - **Security:** Accounts and speaker tools require access control.
  - **Maintainability:** Small team must support system with minimal overhead.

# Applying Architectural Thinking to Lafter

## ❖ Design Decisions Influenced by Characteristics:

- Choose **microservices** for independently deployable features (e.g., joke-posting vs. notifications).
- Use **cloud-based hosting** with autoscaling to manage bursty traffic.
- Implement **CI/CD pipelines** to ensure rapid, reliable deployments.
- Enable **internationalisation** to support a multilingual audience.

## ❖ Examples of Characteristics Applied:

- **Elasticity** → Serverless functions for unpredictable joke-post surges
- **Responsiveness** → Real-time interactions using *WebSockets*
- **Testability** → Unit-tested microservices enable quick fixes

# Architectural Characteristics *vs.* Logical Components

- ❖ **Architectural Characteristics:** How the system performs under various constraints
- ❖ **Logical Components:** What the software does (domain behavior)
- ❖ Both are essential for structural design

*Component:* User Registration

*Characteristic:* Scalability  
(handles thousands of concurrent signups)

*Component:* Content Posting

*Characteristic:* Availability  
(system is up when users post jokes)

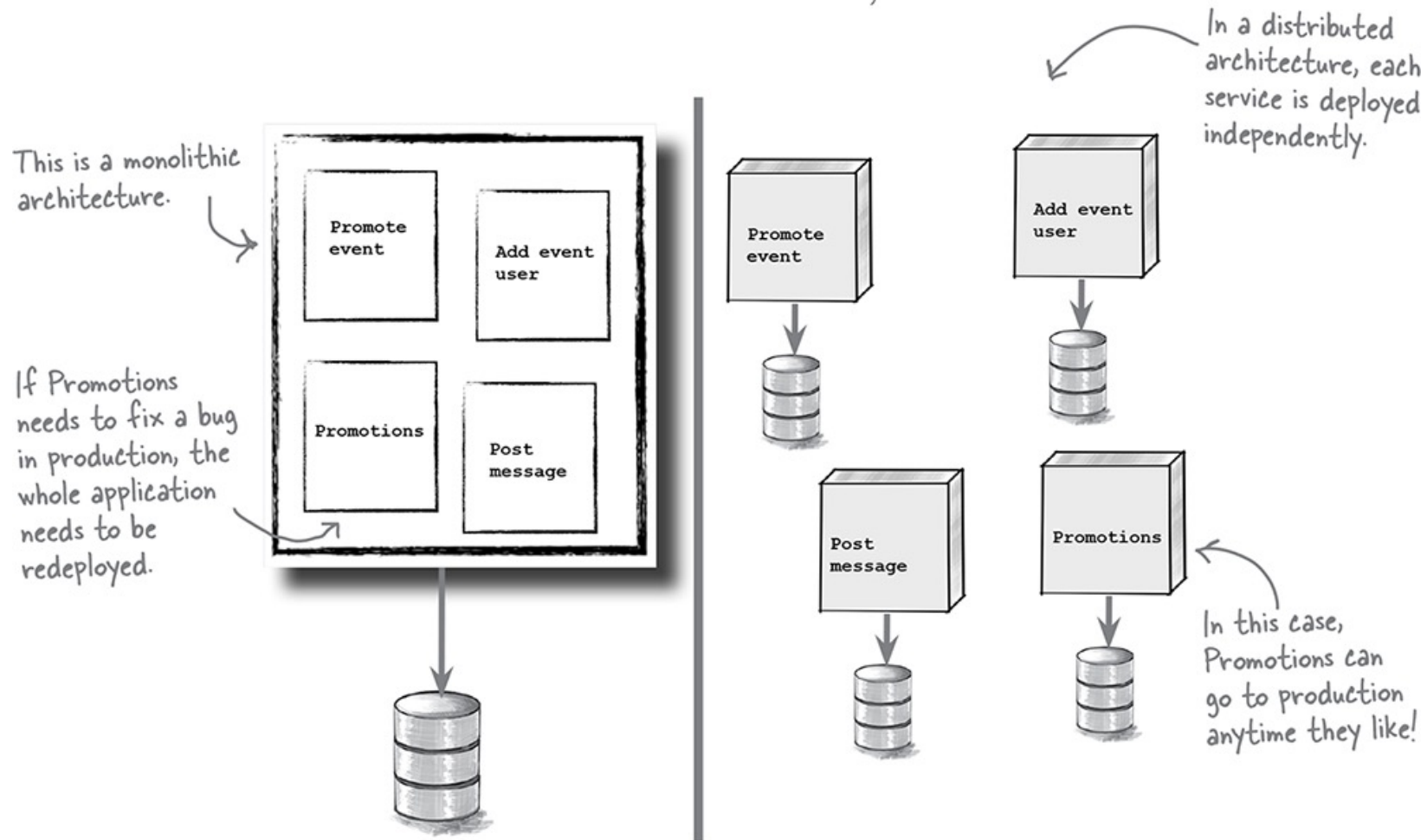
*Component:* Notifications

*Characteristic:* Responsiveness  
(sends real-time updates with low latency)

*Component:* Admin Dashboard

*Characteristic:* Security  
(restricts access to authorized users only)

# Characteristics Affect Structure

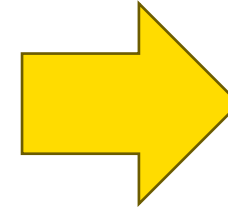




# Characteristics Affect Structure

## Architectural Characteristics

- **Security**: May require encryption layers, role-based access control, audit trails
- **Scalability**: May need load balancers, stateless services, database sharding
- **Availability**: May require failover mechanisms, replication, redundant systems
- **Responsiveness**: Might demand caching, asynchronous processing
- **Elasticity**: May need autoscaling and container orchestration



## Architectural Impact

- **Monolithic vs Microservices**
- **Cloud vs On-Premise deployment**
- Etc.

# Don't Overengineer!

❖ Too many characteristics = complexity

❖ They are:

- Synergistic (affect each other, improving *security* may result in low *performance*)
- Continuously evolving
- Impossible to standardize (*performance* and *responsiveness* might indicate the same behavior)

❖ Limit characteristics to prevent overengineering

- Identifying which characteristics are most critical acts as a filtering mechanism
- Helps eliminate "nice-to-have" features that add unnecessary complexity and cost
- Stay focused on traits essential for success
- Limit to around 7 key characteristics, humans best manage  $7 \pm 2$  items!

# Implicit vs Explicit Characteristics

- ❖ **Explicit:** Stated clearly in the requirements document
  - "The system must support French and Japanese" → Internationalization
  - "Allow only registered users to access admin panel" → Authorization
- ❖ **Implicit:** Not stated, but understood or expected (requires domain/context understanding)
  - Users expect their data to be secure even if not mentioned → Security
  - An app must perform well during high traffic without explicit mention → Performance/Scalability
  - A public website is expected to be available 24/7 → High Availability
- ❖ Architects must **read between** the lines to uncover hidden requirements

## **Explicit**

- Packages are stacked outside a door.



## **Implicit**

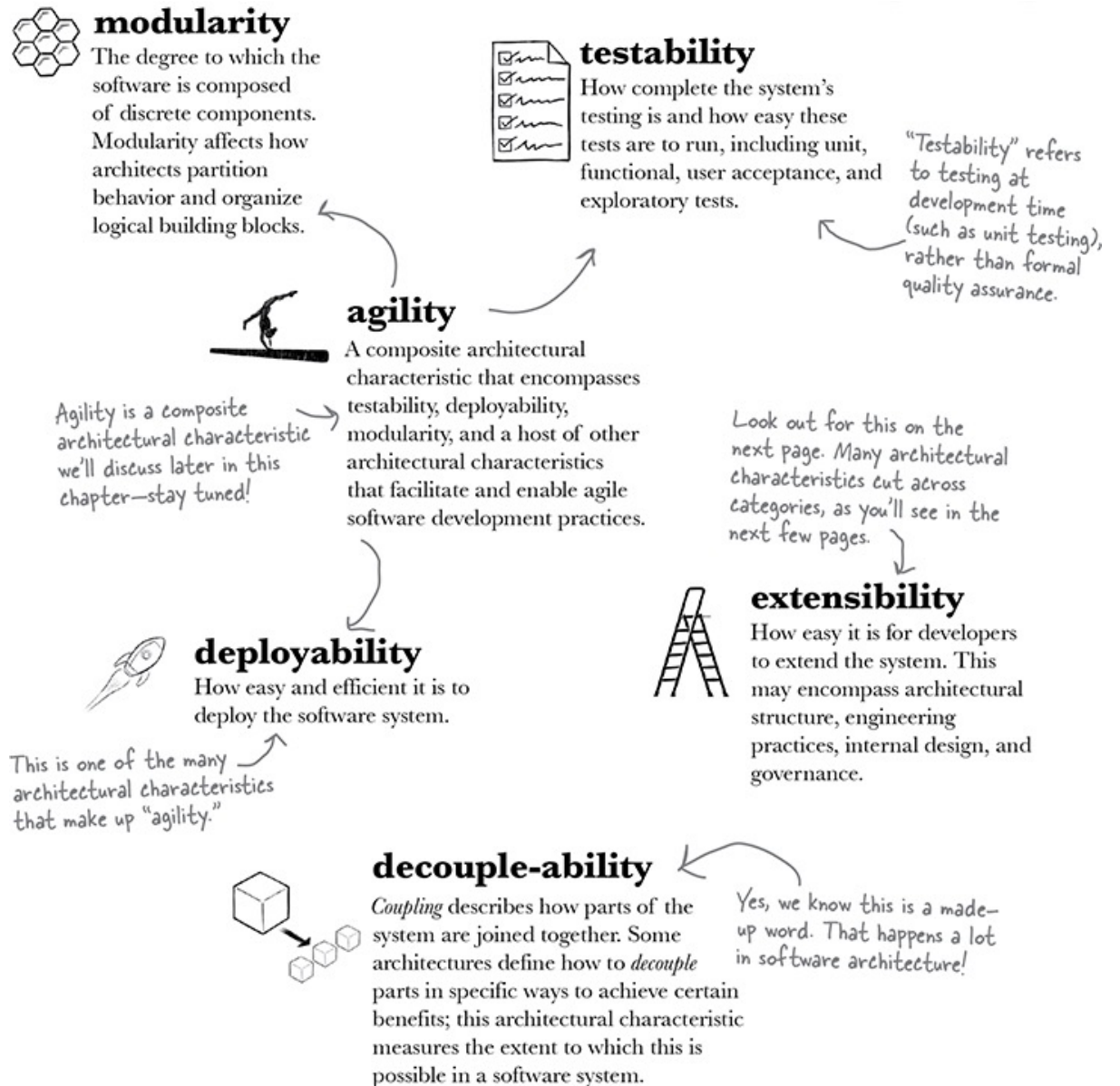
- No one is home.
- This family orders a bunch of stuff online.
- Is this family on vacation?

# Types of Characteristics

- ❖ **Process** Characteristics: Deployability, Maintainability
- ❖ **Structural** Characteristics: Modularity, Coupling
- ❖ **Operational** Characteristics: Scalability, Availability
- ❖ **Cross-cutting** Characteristics: Accessibility, Security

# Process Characteristics

- ❖ Represent the intersection between architecture and the **software development** process
- ❖ Reflect how the **system is built**, tested, deployed, and evolved
- ❖ **Guide decisions** related to engineering practices, automation, and team workflows



# Structural Characteristics

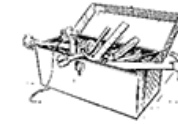
- ❖ Concerned with the **internal structure** and **composition** of the system
- ❖ Influence how **components** are coupled, **interact**, and evolve independently
- ❖ Impact design qualities like **modularity, cohesion**, and **adaptability**



## security

How secure the system is, holistically. Does the data need to be encrypted in the database? How about for network communication between internal systems? What type of authentication needs to be in place for remote user access?

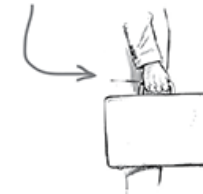
Security appears in every application, as an implicit or explicit architectural characteristic.



## maintainability

How easy it is for architects and developers to apply changes to enhance the system and/or fix bugs.

Portability can apply to any part of the system, including the user interface and implementation platform.



## portability

How easy it is to run the system on more than one platform (for example, Windows and macOS).

This is one of those characteristics that belong to more than one category.



## extensibility

How easy it is for developers to extend the system. This may encompass architectural structure, engineering practices, internal design, and governance.

Some architectural characteristics cover development concerns rather than purely domain concerns.



## localization

How well the system supports multiple languages, units of measurement, currencies, and other factors that allow it to be used globally.

Another flavor of localization is internationalization (i18n).

# Operational Characteristics

- ❖ Represent how architectural decisions shape and support system **behavior at runtime**
- ❖ Define what the operations team can **monitor, control, or adapt** while the system is running
- ❖ Directly influence system **reliability, performance, and fault tolerance**



## availability

What percentage of the time the system needs to be available and, if 24/7, how easy it is to get the system up and running quickly after a failure.

Usually represented as a number of "nines" (99.999% uptime = 5 nines, a bit under 6 minutes/year).



## recoverability

How quickly the system can get online again and maintain business continuity in case of a disaster. This will affect the backup strategy and requirements for duplicated hardware.

A good example of the axiom that you can take any adjective and add "-ility" to make a new architectural characteristic!



## robustness

The system's ability to handle errors and boundary conditions while running, such as if the power, internet connection, or hardware fails.



## performance

How well the system achieves its timing requirements using the available resources.

As you will see shortly, "performance" has many different aspects.

When these are important, they are very important.



## reliability/safety

Whether the system needs to be fail-safe, or if it is mission critical in a way that affects lives. If it fails, will it endanger people's lives or cost the company large sums of money? Common for medical systems, hospital software, and airplane applications.

Some "-ilities" are easier to achieve than others. This one is often difficult.



## scalability

How well the system performs and operates as the number of users or requests increases.

Our Laffer application will definitely need this!



# Cross-Cutting Characteristics

- ❖ Span **multiple parts** of the system and affect other characteristics
- ❖ Often enforced through design, tooling, and governance





# Composite Characteristics

- ❖ Formed from **multiple** simpler traits
- ❖ These high-level characteristics reflect complex system qualities that require multiple dimensions to evaluate.
- ❖ Examples:
  - **Reliability** = Availability + Consistency + Data Integrity
  - **Resilience** = Robustness + Fault Tolerance + Recoverability
- ❖ Must **break down** into measurable parts

# Sources of Characteristics

- ❖ **Problem Domain (Feature-Driven)**: Driven by product goals, system features, and expected usage patterns.
  - A real-time multiplayer game must be highly responsive and scalable due to concurrent users → Responsiveness, Scalability
  - An e-commerce site must support flash sales and high traffic → Elasticity, Performance, Availability
- ❖ **Environmental Awareness (Organizational Constraints)**: Driven by company's culture, budget, and capabilities
  - A startup must deliver features fast → favors Deployability, Agility
  - A globally distributed team → needs Testability, Modularity for asynchronous collaboration
  - Legacy-heavy organizations → may prioritize Integrability with existing systems

# Sources of Characteristics

❖ **Holistic Domain Knowledge (Industry and Compliance Expectations):** Driven by regulatory standards, industry best practices, and user trust factors.

- A banking app must comply with regulations → Security, Auditability, Availability
- A healthcare platform must protect patient data → Confidentiality, Data Integrity, Compliance
- Government services need to meet accessibility and privacy laws → Accessibility, Security, Maintainability

❖ **Why It Matters:**

- Ignoring a source can lead to critical failures later
- Architects must triangulate across all three to identify the most important characteristics

# Translating Business Goals into Architecture

- ❖ A core responsibility of the architect is to **translate** high-level business goals into concrete architectural characteristics and decisions.

## Examples:

- ❖ Business: “**The system must always work.**”
  - → High Availability, Fault Tolerance, Robustness
- ❖ Business: “**Customers shouldn’t wait.**”
  - → Responsiveness, Performance, Latency Budgeting
- ❖ Business: “**We need to move fast and innovate.**”
  - → Deployability, Modularity, Testability
- ❖ Business: “**We need to meet compliance and regulation.**”
  - → Security, Auditability, Traceability, Accessibility

# Trade-offs Between Architectural Characteristics (1)

- ❖ Architectural characteristics frequently **compete** or **conflict**
- ❖ Enhancing one trait can reduce or compromise another

## Examples:

- **Security vs. Performance** [More security controls (e.g., encryption, validation) add processing overhead]
- **Scalability vs. Simplicity** [Scalable systems often introduce distributed complexity (e.g., microservices, load balancers)]
- **Availability vs. Maintainability** [High availability may require complex failover and redundancy, complicating upgrades and maintenance]

## Key Insight:

- Trade-offs are **not flaws**, but **conscious** architectural **decisions**
- There are **no universally right** answers—*"It depends."*

# Trade-offs Between Architectural Characteristics (2)

## More Examples:

- ❖ **Deployability** vs. **Robustness**: Rapid deployments can reduce test cycles and stability
- ❖ **Responsiveness** vs. **Data Consistency**: Fast user interactions might rely on eventually consistent models
- ❖ **Flexibility** vs. **Performance**: Designing for plugin support or configuration often introduces performance bottlenecks

## Best Practices:

- ❖ **Engage stakeholders** early to understand what matters most
- ❖ **Prioritise** characteristics based on system goals, user needs, and domain constraints
- ❖ **Trade-offs** reflect organizational priorities and constraints
- ❖ Use **Architectural Decision Records (ADRs)** to document trade-offs and rationale

# Summary

- ❖ Architecture is about making **conscious trade-offs**
  - ❖ Every decision comes with **upsides** and **downsides**
  - ❖ Capture both the decision and the **reason behind** it
  - ❖ Embrace change—**architecture evolves** with the system
- 
- ❖ Architecture isn't about right answers—it's about **right reasoning**