Architectural Patterns

We discussed about patterns for implementing business logic in the tactical design documents, now we are going to talk about design decisions in a broader context: the different ways to orchestrate the interactions and dependencies between a system’s components.

# Business Logic vs Architectural Patterns

**The variety of concerns** **that a codebase has to take care of makes it easy for its business logic to become diffused among the different components**: that is, for some of the logic to be implemented in the user interface or database, or be duplicated in different components.

**Lacking strict organization** in implementation concerns makes the codebase hard to change**:**

* **When the business logic has to change, it may not be evident what parts of the codebase have to be affected by the change**.
* The change may have unexpected effects on seemingly unrelated parts of the system.
* Conversely, it may be easy to miss code that has to be modified. All of these issues dramatically increase the cost of maintaining the codebase.

**Architectural patterns introduce organizational principles for the different aspects of**

**a codebase and present clear boundaries between them**:

**how the business logic is wired to the system’s input, output, and other infrastructural components.**

This affects how these components interact with each other: what knowledge they share

and how the components reference each other.

Choosing the appropriate way to organize the codebase, or the correct architectural

pattern, is crucial to support implementation of the business logic in the short term

and alleviate maintenance in the long term.

Let’s explore three predominant application architecture patterns and their use cases: layered architecture, ports & adapters, and CQRS.

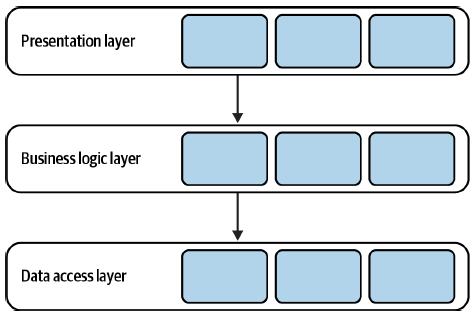
# Layered Architecture

Layered architecture is one of the most common architectural patterns. It organizes

the codebase into horizontal layers, with each layer addressing one of the following

technical concerns**: interaction with the consumers, implementing business logic, and**

**persisting the data.** You can see this represented in Figure 8-1.



In its classic form, the layered architecture consists of three layers: the presentation

layer (PL), the business logic layer (BLL), and the data access layer (DAL).

## Presentation Layer

The presentation layer, shown in Figure 8-2, implements the program’s user interface

**for interactions with its consumers**. In the pattern’s original form, this layer denotes a

graphical interface, such as a web interface or a desktop application.

In modern systems, however, the presentation layer has a broader scope: that is, **all**

**means for triggering the program’s behavior, both synchronous and asynchronous**.( and I add : the means to exposing some output??)

For example:

• Graphical user interface (GUI)

• Command-line interface (CLI)

• API for programmatic integration with other systems

• **Subscription to events in a message broker**

• **Message topics for publishing outgoing events**

**All of these are the means for the system to receive requests from the external environment and communicate the output. Strictly speaking, the presentation layer is the Program’s public interface.**

## Business Logic Layer

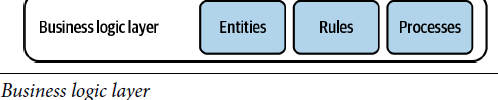
As the name suggests, this layer is responsible **for implementing and encapsulating**

**the program’s business logic.** This is the place where business decisions are implemented.

As Eric Evans says,1 this layer is the heart of software.

This layer is where the business logic patterns described in Chapters 5–7 are implemented—

**for example,** **active records or a domain model** (see Figure 8-3).



## Data Access Layer

The data access layer **provides access to persistence mechanisms**. In the pattern’s original form, this referred to the system’s database. However, as in the case of the presentation layer, the layer’s responsibility is **broader for modern systems.**

First, ever since the NoSQL revolution broke out, it is common for a system to work

with multiple databases. For example, a document store can act as the operational

database, a search index for dynamic queries, and an in-memory database for

performance-optimized operations.

Page 120…..