


# Design Your Software Architecture Using Industry-Standard Patterns

4 hours  Hard

Last updated on 6/29/20



## Layered Architecture

 [Log in](#) or [subscribe](#) for free to enjoy all this course has to offer!

In this chapter, we are going to cover the layered architecture, what it is, and in what cases it is the perfect solution for a design problem. At the end of the chapter, I will show you a real-world example (Amaze project management software), and then you will solve a similar case applying what you've learned.

### What Is Layered Architecture?



Have you ever wondered how Google makes Gmail work in different languages all over the world? Users can use Gmail every day in English, Spanish, French, Russian, and many more languages.

Did Google develop different Gmail applications for each country? Of course not. They developed an internal version that does all the message processing, and then developed different external user interfaces that work in many languages.

Google developed the Gmail application in **different layers**:

- There is an internal layer that does all the processing.
- There is an external layer that communicates with the users in their language.
- There is also another layer that interacts with a database where user email messages are stored (millions or maybe billions).

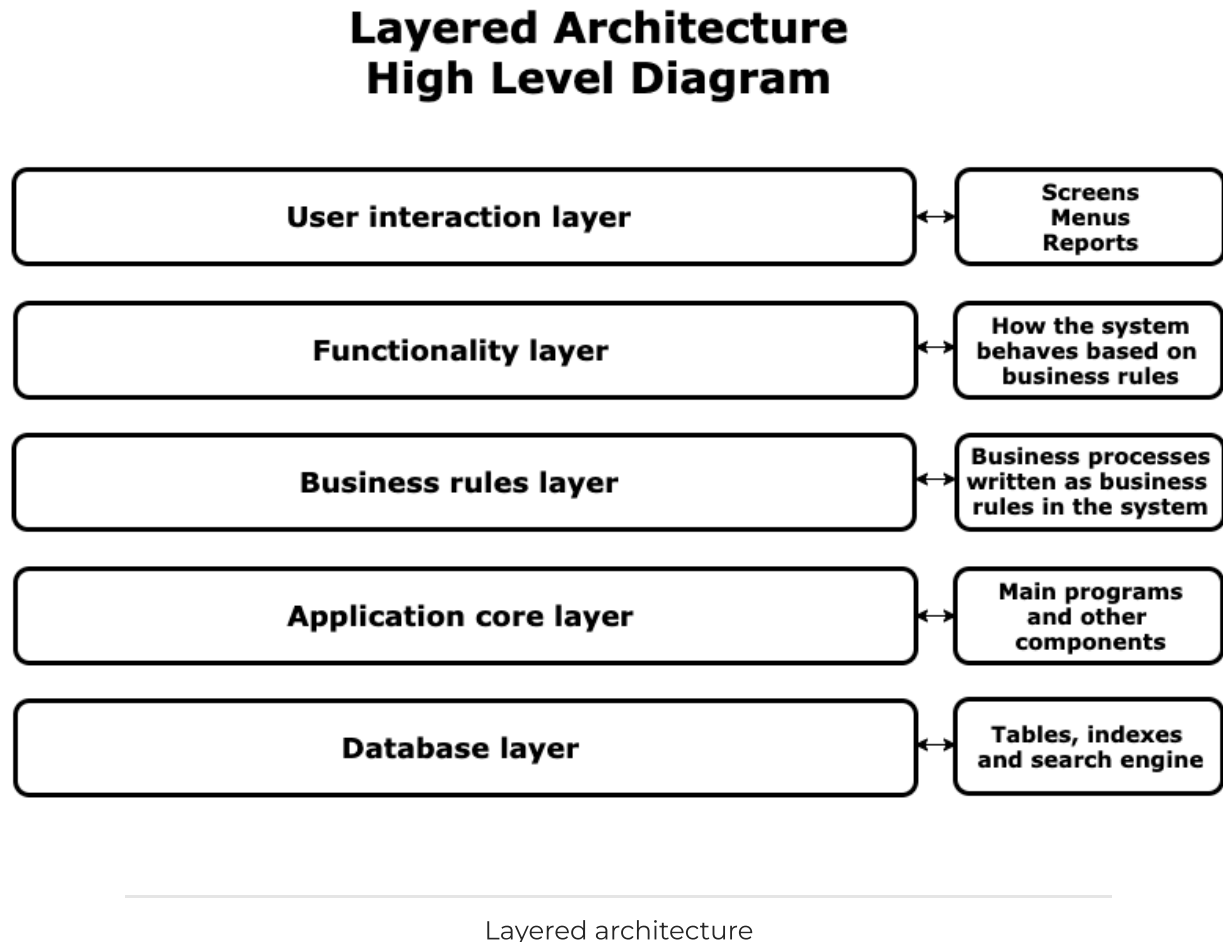
Gmail is divided into at least three layers, every one of them has a mission, and they exist separately to handle different processes at different levels. It is an excellent example of a layered

architecture.

## What Is the Structure of a Layered Architecture?



Let's see what this looks like:



As you can see in the diagram above, a standard layered architecture has five parts:

- **User interaction layer:** This is the layer that interacts with users through screens, forms, menus, reports, etc. It is the most visible layer of the application. It defines how the application looks.
- **Functionality layer:** This is the layer that presents the functions, methods, and procedures of the system based on the business rules layer. It determines how the pull-down menus work, how the buttons work, and how the system navigates through screens.
- **Business rules layer:** This layer contains rules that determine the behavior of the whole application, such as, "If an invoice is printed, then send an email to the customer, select all items sold, and decrease their stock in the stock management module."
- **Application core layer:** This server contains the main programs, code definitions, and basic functions of the application. Programmers work in this layer most of the time.
- **Database layer:** This layer contains the tables, indexes, and data managed by the application. Searches and insert/delete/update operations are executed here.

How does this work in practice?

- An ERP System (accounts payable, accounts receivable, stock management, HR management, production management, provider management, purchasing, treasury, finance, accounting, etc.) has a user interaction layer for each module: screens, forms, menus, reports. This is what the user sees and what they use.
- The functionality layer navigates through the different modules, presents screen sequences to the user, and does all data input operations.
- The business rules layer determines the behavior of the modules of the ERP: “If a new employee is entered into the HR and payroll modules, then insert an introductory course in the employee's training menu.”
- The application core layer is the place where all system code is located. This is where developers add customizations and new functionalities.
- The database layer contains the tables, indexes, and data managed by each of the modules.

There are several **advantages** to using layered architecture:

- Layers are autonomous: A group of changes in one layer does not affect the others. This is good because we can increase the functionality of a layer, for example, making an application that works only on PCs to work on phones and tablets, without having to rewrite the whole application.
- Layers allow better system customization.

There are also a few key **disadvantages**:

- Layers make an application more difficult to maintain. Each change requires analysis.
- Layers may affect application performance because they create overhead in execution: each layer in the upper levels must connect to those in the lower levels for each operation in the system.

Not all applications are worth the additional maintenance required for layered architecture.

When Would I Use Layered Architecture?

▼

Here are situations that this would be useful for:

Example Name	Definition	Real-World Example	Advantages
Control system for self-	A control system for self-driving vehicles connects the different parts of the vehicle to	• Tesla Control Module (tesla.com).	Control systems for self-driving cars use a multi-layered architecture: the same

driving vehicles	a control module that corrects the direction of the driving, speed, and trajectory.	<ul style="list-style-type: none"> <li>Waymo software (waymo.com).</li> </ul>	system can be used if some parts of the car change with time, or can be used for different models of cars. Only one layer is modified.
Home banking website	A home banking website is a system that allows bank customers to manage their accounts using the internet. This system is usually connected to the bank's core system.	Any bank has a home banking system today.	These systems are usually organized in layers, so if the bank wants the home banking system to look different (due to a new branding campaign, for example) no internal systems have to be changed.

## Case-Study: Solving a Business Problem With Layered Architecture



Let's see how a layered architecture was used to solve a real problem.

Amaze is a project management software company. Their product is sold globally with a monthly pay-per-user model and widely known among the project management community for being easy to use and able to operate on many different devices (PCs, Notebooks, laptops, tablets, iPhones, iPads, and Android phones).

### What's the Business Problem?

The business problem is very straightforward: Amaze must work on any popular device on the market and be able to support future devices. There must be only one version of the software for all devices. No special cases, no exceptions allowed.

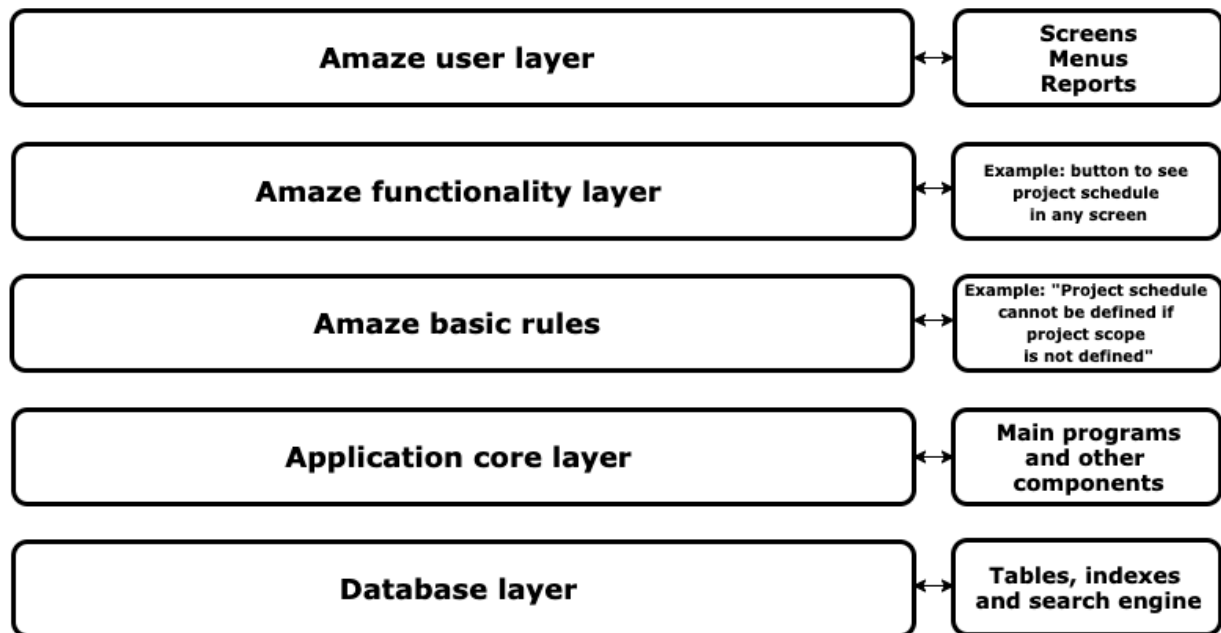
So to sum up:

- We know that users have different devices.
- There must be only one software application because the company wants to have low software maintenance costs.
- When new device launches, we do not want to change the whole software product.

### What's the Solution?

Now that we've walked through the thought process, let's see the detailed architecture diagram:

## Layered Architecture Amaze



Amaze layered architecture

In our case, the **Amaze basic rules layer** is critical. This layer contains rules that determine the behavior of the whole application, such as, "You can create a project schedule only if the project scope is defined." The product's intelligence is in this layer: all special features that come from decades of experience in projects are developed there. The **application core layer** will be the most significant part of the application code.

### Try it out for yourself!

Now that you've seen one solution see if you can apply what you've learned to another!

**Context:** You have been a software architect at a major insurance company in your country for the last four years. Your boss is Carla, the chief information officer (CIO) of the company. The company has offices in 24 cities and more than 2,100 employees.

**Your Mission:** You have been asked to develop a software system for managing new insurance policies for your clients' staff in different industries (finance, manufacturing, technology, engineering, etc.). Each industry has different customs and requirements for managing employee benefits.

Here are some **key questions** you should ask yourself:

1. How are you going to handle different industry requirements in one single system?

#

2. What do all policies in all industries have in common, regardless of the industry, customer, or insured goods?
3. How could you separate common requirements for all industries and specific requirements for each industry?
4. Who is going to define the business rules for each industry?

Can you produce an architecture diagram for this business setting?

Once you've written out your version, check it against my **solution**.

## Let's Recap!



Architecture Model	Description	Advantages	Disadvantages	When to use it
<b>Layered</b>	Software operates in layers that allow each component to be independent of the rest.	Encapsulation of hardware, software, and functionality.  If a layer is changed, the rest of the layers stay the same.	For small applications, many layers create a performance problem and are very difficult to maintain.	Only for big applications.



**PLUG-IN ARCHITECTURE**

**DATA-CENTERED ARCHITECTURE**



## Teacher

### José Esterkin

Software engineer, project manager and trainer. Director of Positive, a project management consulting firm based in Buenos Aires.



OPPORTUNITIES

▼

SUPPORT

▼

FOR BUSINESS

▼

MORE

▼

 English

▼



