Software Architecture - The Difference Between Architecture and Design

Mohamed Aladdin Follow
Jul 27, 2018 · 7 min read ★

Many people don't really know the difference between software architecture and software design. Even for developers, the line is often blurry and they might mix up elements of software architecture patterns and design patterns. As a developer myself, I would like to simplify these concepts and explain the differences between software design and software architecture. In addition, I will show you why it is important for a developer to know a little bit about software architecture and a lot of software design. So, let's start.

The Definition of Software Architecture

In simple words, software architecture is the process of converting software characteristics such as flexibility, scalability, feasibility, reusability, and security into a structured solution that meets the technical and the business expectations. This definition leads us to ask about the characteristics of a software that can affect a software architecture design. There is a long list of characteristics which mainly represent the business or the operational requirements, in addition to the technical requirements.

The Characteristics of Software Architecture

As explained, software characteristics describe the requirements and the expectations of a software in operational and technical levels. Thus, when a product owner says they are competing in a rapidly changing markets, and they should adapt their business model quickly. The software should be "**extendable, modular and maintainable**" if a business deals with urgent requests that need to be completed successfully in the matter of time. As a software architect, you should note that the **performance** and **low fault tolerance, scalability and reliability** are your key characteristics. Now, after defining the previous characteristics the business owner tells you that they have a limited budget for that project, another characteristic comes up here which is "**the feasibility.**"

Here you can find a full list of software characteristics, also known as "quality attributes," here.

Software Architecture Patterns

Most people have probably heard of the term "**MicroServices**" before. **MicroServices** is one of many other software architecture patterns such as Layered Pattern, Event-Driven Pattern, Serverless Pattern and many more. Some of them will be discussed later in this article. The Microservices pattern received its reputation after being adopted by Amazon and Netflix and showing its great impact. Now, let's dig deeper into the architecture patterns.

**A quick note, please don't mix up design patterns like Factory or adaptor patterns and the architecture patterns. I will discuss them later.

Serverless Architecture

This element refers to the application solution that depends on third-party services to manage the complexity of the servers and backend management. Serverless Architecture is divided into two main categories. The first is "Backend as a service (BaaS)" and the second is "Functions as a Service (FaaS)." The serverless architecture will help you save a lot of time taking care and fixing bugs of deployment and servers regular tasks.
The most famous provider for serverless API is Amazon AWS "Lambda."

You can read more about this here.

Event-Driven Architecture

This architecture depends on Event Producers and Event Consumers. The main idea is to decouple your system's parts and each part will be triggered when an interesting event from another part has got triggered. Is it complicated? Let's simplify it. Assume you design an online store system and it has two parts. A purchase module and a vendor module. If a customer makes a purchase, the purchase module would generate an event of "orderPending" Since the vendor module is interesting in the "orderPending" event, it will be listening, in case one is triggered. Once the vendor module gets this event, it will execute some tasks or maybe fire another event for order more of the product from a certain vendor.

Just remember the event-producer does not know which event-consumer listening to which event. Also, other consumers do not know which of them listens to which events. Therefore, the main idea is decoupling the parts of the system.

If you are interested in learning more about this, click here.

Microservices Architecture

Microservices architecture has become the most popular architecture in the last few years. It depends on developing small, independent modular services where each service solves a specific problem or performs a unique task and these modules communicate with each other through well-defined API to serve the business goal. I do not have to explain more just look at this image.

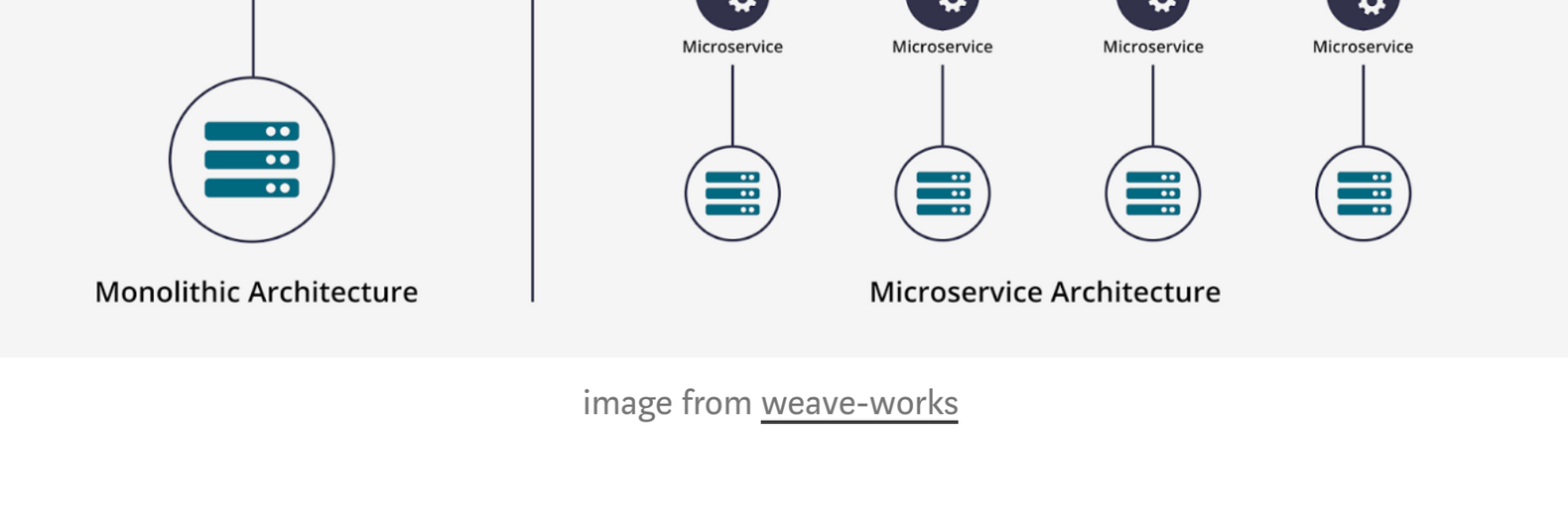

image from weave-works

Software Design

While software architecture is responsible for the skeleton and the high-level infrastructure of a software, the software design is responsible for the code level design such as, what each module is doing, the classes scope, and the functions purposes, etc.

If you are a developer, it is important for you to know what the SOLID principle is and how a design pattern should solve regular problems.

SOLID refers to **S**ingle Responsibility, **O**pen Closed, **L**iskov substitution, **I**nterface Segregation and **D**ependency Inversion Principles.

- **Single Responsibility Principle** means that each class has to have one single purpose, a responsibility and a reason to change.
- **Open Closed Principle**: a class should be open for extension, but closed for modification. In simple words, you should be able to add more functionality to the class but do not edit current functions in a way that breaks existing code that uses it.
- **Liskov substitution principle**: this principle guides the developer to use inheritance in a way that will not break the application logic at any point. Thus, if a child class called "XyClass" inherits from a parent class "AbClass", the child class shall not replicate a functionality of the parent class in a way that change the behavior parent class. So you can easily use the object of XyClass instead of the object of AbClass without breaking the application logic.
- **Interface Segregation Principle**: Simply, since a class can implement multiple interfaces, then structure your code in a way that a class will never be forced to implement a function that is not important to its purpose. So, categorize your interfaces.
- **Dependency Inversion Principle**: If you ever followed TDD for your application development, then you know how decoupling your code is important for testability and modularity. In other words, If a certain Class "ex: Purchase" depends on "Users" Class then the User object instantiation should come from outside the "Purchase" class.

Design Patterns

- **Factory Pattern**: it is the most used design pattern in the OOP world because it saves a lot of time in the future when you have to modify one of the classes you used. Look at this example:

Imagine you want to instantiate a Users() Model Class, there are two ways to do it:

1 — $users = new Users();
2 — $users = DataFactory::get('Users');

```
class DataFactory
{
    public static function get($model){
        switch($model){
            case 'Users':
                return new Users();
            case 'Purchase':
                return new Purchase();
            default:
                return new $className();
                break;
        }
    }
}
```

I would prefer the second way for two reasons among several ones. First, changing class name from "Users" to "UsersData" will only require one change in one place "inside the data factory" and the rest of your code will be the same. Second, if the class Users start taking parameters like Users($connection); then you also will need to change it in one place not in every function that required Users object. So, if you think the first way is better, think again.

- **Adapter Pattern:** Adapter Pattern is one of the structural design patterns. From its name, you would expect that it converts the unexpected usage of class to an expected one.

Imagine that your application deals with Youtube API and in order to get access token, you have to call a function called getYoutubeToken();

```
// Youtube version 1 By Youtube
class Youtube {

    public function getYoutubeToken() {
        // Youtube code here
    }
}
```

So, you called this function in 20 different places in your application.

```
$token = $youtube->getYoutubeToken();
```

Then, Google releases new version of Youtube API and they renamed it to getAccessToken();

```
// Youtube version 2 By Youtube
class Youtube {

    public function getAccessToken() {
        // Youtube code here
    }
}
```

Now you will have to find and replace the function name everywhere across your application or you can create an Adapter class like the following example:

```
// You create this class to contain future problems
class YoutubeAdapter{

    // Function by you
    public function getToken(){
        $youtube = new Youtube();
        $token = $youtube->getYoutubeToken();

        return $token;
    }
}
```

```
$token = $youtubeAdapter->getToken();
```

In this case, you only have to change one line and the rest of your application will keep working as usual.

```
// You create this class to contain future problems
class YoutubeAdapter{

    // Function by you
    public function getToken(){
        $youtube = new Youtube();
        //$token = $youtube->getYoutubeToken();
        $token = $youtube->getAccessToken();

        return $token;
    }
}
```

Since this article does not talk about design patterns in detail, here are some helpful links if you want to learn more:

https://code.tutsplus.com/series/design-patterns-in-php--cms-747
http://www.phptherightway.com/pages/Design-Patterns.html

Remember there is a difference between a software architect and a software developer. Software architects have usually experienced team leaders, who have good knowledge about existing solutions which help them make right decisions in the planning phase. A software developer should know more about software design and enough about software architecture to make internal communication easier within the team.

Become An Expert Developer with These Advanced Coding Tips (part 1)
The worst thing that can happen to a developer is to feel stuck in his current level of skills. I mean when you know…
codeburst.io

More to Read:
- *Advanced Coding Skills, Techniques and Ideas*
- *Become An Expert Developer with These Advanced Coding Tips (part 1)*
- *Software Architecture: Architect Your Application with AWS*

WRITTEN BY
Mohamed Aladdin Follow
Software engineer and Entrepreneur, love to learn a lot and spread the knowledge. https://articles.speakytext.com