Copilot

Sure, in software engineering, “design principles” are a set of guidelines that helps developers to make a good system design. They promote code readability, reusability, and maintainability. Here are some of the most common design principles with examples:

1. **Single Responsibility Principle (SRP)**: A class should have only one reason to change. This means a class should only have one job or responsibility.

```
# Good
class User:
    def get_user(self, id):
        pass

class UserDB:
    def save_user(self, user):
        pass

# Bad
class User:
    def get_user(self, id):
        pass

    def save_user(self, user):
        pass
```

1. **Open-Closed Principle (OCP)**: Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification.

```
# Good
class Discount:
    def __init__(self, customer, price):
        self.customer = customer
        self.price = price

    def give_discount(self):
        return self.price

class VIPDiscount(Discount):
    def give_discount(self):
        return super().give_discount() * 0.8

# Bad
class Discount:
    def __init__(self, customer, price, customer_type):
        self.customer = customer
        self.price = price
        self.type = customer_type

    def give_discount(self):
        if self.type == 'VIP':
            return self.price * 0.8
        if self.type == 'Normal':
            return self.price
```

1. **Liskov Substitution Principle (LSP)**: Subtypes must be substitutable for their base types.

```
# Good
class Bird:
    def fly(self):
        pass

class Duck(Bird):
    def fly(self):
        pass

class Ostrich(Bird):
    def fly(self):
```

```
        raise NotImplementedError("Ostrich can't fly.")

# Bad
class Bird:
    def fly(self):
        pass

class Ostrich(Bird):
    pass

ostrich = Ostrich()
ostrich.fly()  # Ostrich can't fly but we are making it fly.
```

1. **Interface Segregation Principle (ISP)**: Clients should not be forced to depend on interfaces they do not use.

```
# Good
class Worker:
    def work(self):
        pass

class Eater:
    def eat(self):
        pass

class Human(Worker, Eater):
    def work(self):
        pass

    def eat(self):
        pass

# Bad
class Worker:
    def work(self):
        pass

    def eat(self):
        pass

class Robot(Worker):
    def work(self):
        pass

    def eat(self):
        pass  # Robots do not eat, but we have to implement this method.
```

1. **Dependency Inversion Principle (DIP)**: High-level modules should not depend on low-level modules. Both should depend on abstractions.

```
# Good
class LightBulb:
    def turn_on(self):
        pass

    def turn_off(self):
        pass

class ElectricPowerSwitch:
    def __init__(self, l: LightBulb):
        self.lightBulb = l

    def press(self):
        if self.lightBulb.on:
            self.lightBulb.turn_off()
        else:
            self.lightBulb.turn_on()

# Bad
class LightBulb:
    def turn_on(self):
        pass
```

```
    def turn_off(self):
        pass

class ElectricPowerSwitch:
    def __init__(self):
        self.lightBulb = LightBulb()  # Depends on a low-level module directly

    def press(self):
        if self.lightBulb.on:
            self.lightBulb.turn_off()
        else:
            self.lightBulb.turn_on()
```

These principles are part of the SOLID principles, a popular set of design principles in object-oriented programming. They can help you write code that is easier to understand, maintain, and extend. However, like all principles, they are guidelines, not hard and fast rules. It's important to understand the principles and the benefits they can provide, but also to use your judgment about when to apply them.