## Copilot

Absolutely, I'd be happy to explain the SOLID principles in more detail. SOLID is an acronym for five design principles intended to make software designs more understandable, flexible, and maintainable. Here's a more detailed explanation of each principle:

1. **Single Responsibility Principle (SRP)**: This principle states that a class should have only one reason to change. It means that a class should only have one job or responsibility. If a class has more than one responsibility, it becomes coupled. A change to one responsibility results in modification of the other responsibility.

2. **Open-Closed Principle (OCP)**: According to this principle, "software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification". What it means is that you should be able to add new functionality or modify existing behavior without changing the existing code structure or causing side effects.

3. **Liskov Substitution Principle (LSP)**: This principle states that "objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program". In other words, your derived classes must be substitutable for their base classes without causing issues.

4. **Interface Segregation Principle (ISP)**: This principle states that "clients should not be forced to depend on interfaces they do not use". This means that a class should not have to implement methods it doesn't use. Instead of one fat interface, numerous small interfaces are preferred based on groups of methods, each one serving one submodule.

5. **Dependency Inversion Principle (DIP)**: This principle states that "high-level modules should not depend on low-level modules. Both should depend on abstractions". In other words, dependencies on the source code should refer to abstractions, not on concretions. It allows for decoupling.

These principles, when combined together, make it easy for a programmer to develop software that is easy to maintain and extend over time. They also make it easy for developers to avoid code smells, easily refactor code, and are also a part of the agile or adaptive software development.