SOLID

In object-oriented programming, **SOLID** is a mnemonic acronym for five principles intended to make source code more understandable, flexible, and maintainable. Although the principles apply to object-oriented programming, they can also form a core philosophy for methodologies such as agile software development and adaptive software development.[1]

Software engineer and instructor Robert C. Martin[2][3][1] introduced the basic principles of SOLID design in his 2000 paper *Design Principles and Design Patterns* about software rot.[3][4]:2–3 The *SOLID* acronym was coined around 2004 by Michael Feathers.[5]

Principles

Single responsibility principle

The single-responsibility principle (SRP) states that there should never be more than one reason for a class to change.[6] In other words, every class should have only one responsibility.[7]

Importance:

- Maintainability: When classes have a single, well-defined responsibility, they're easier to understand and modify.
- Testability: It's easier to write unit tests for classes with a single focus.
- Flexibility: Changes to one responsibility don't affect unrelated parts of the system.[7]

Open–closed principle

The open–closed principle (OCP) states that software entities should be open for extension, but closed for modification.[8]

Importance:

- Extensibility: New features can be added without modifying existing code.
- Stability: Reduces the risk of introducing bugs when making changes.
- Flexibility: Adapts to changing requirements more easily.

Liskov substitution principle

The Liskov substitution principle (LSP) states that functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it.[9] [9]

Importance:

- Polymorphism: Enables the use of polymorphic behavior, making code more flexible and reusable.
- Reliability: Ensures that subclasses adhere to the contract defined by the superclass.
- Predictability: Guarantees that replacing a superclass object with a subclass object won't break the program.[9]

Interface segregation principle

The interface segregation principle (ISP) states that clients should not be forced to depend upon interfaces that they do not use.[10][4]

Importance:

- Decoupling: Reduces dependencies between classes, making the code more modular and maintainable.
- Flexibility: Allows for more targeted implementations of interfaces.
- Avoids unnecessary dependencies: Clients don't have to depend on methods they don't use.

Dependency inversion principle

The dependency inversion principle (DIP) states to depend upon abstractions, not concretes.[11][4]

Importance:

- Loose coupling: Reduces dependencies between modules, making the code more flexible and easier to test.
- Flexibility: Enables changes to implementations without affecting clients.
- Maintainability: Makes code easier to understand and modify.[11][4]

See also

- Code reuse
- GRASP (object-oriented design)
- Inheritance (object-oriented programming)
- List of software development philosophies

References

1. Metz, Sandi (May 2009). "SOLID Object-Oriented Design". *YouTube*. Archived from the original on 2021-12-21. Retrieved 2019-08-13. Talk given at the 2009 Gotham Ruby Conference.
2. Martin, Robert C. "Principles Of OOD". *ButUncleBob.com*. Archived from the original on Sep 10, 2014. Retrieved 2014-07-17.. (Note the reference to "the first five principles", although the acronym is not used in this article.) Dates back to at least 2003.
3. Martin, Robert C. (13 Feb 2009). "Getting a SOLID start". *Uncle Bob Consulting LLC (Google Sites)*. Archived from the original on Sep 17, 2013. Retrieved 2013-08-19.
4. Martin, Robert C. (2000). "Design Principles and Design Patterns" (PDF). *objectmentor.com*. Archived from the original on 2015-09-06.
5. Martin, Robert (2018). *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Pearson. p. 58. ISBN 978-0-13-449416-6.
6. "Single Responsibility Principle" (PDF). *objectmentor.com*. Archived from the original on 2 February 2015.
7. Martin, Robert C. (2003). *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall. p. 95. ISBN 978-0135974445.
8. "Open/Closed Principle" (PDF). *objectmentor.com*. Archived from the original on 5 September 2015.
9. "Liskov Substitution Principle" (PDF). *objectmentor.com*. Archived from the original on 5 September 2015.

10. "Interface Segregation Principle" (PDF). *objectmentor.com*. 1996. Archived from the original on 5 September 2015.
11. "Dependency Inversion Principle" (PDF). *objectmentor.com*. Archived from the original on 5 September 2015.