SOLID principles

Ethan Kent

Spoonflower

March 23, 2023

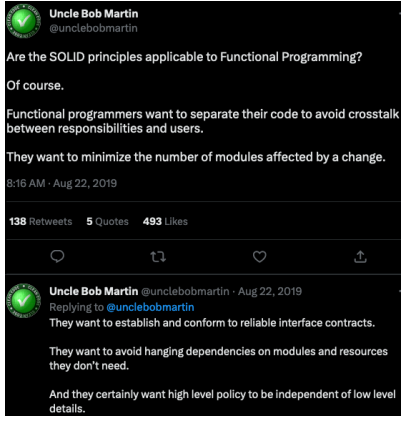
Purpose of the SOLID principles

The SOLID principles tell us how to arrange our functions and data structures into . . . grouping[s] of functions and data. The goal of the principles is the creation of mid-level software structures that:

- *Tolerate change,*
- *Are easy to understand, and*
- *Are the basis of components that can be used in many software systems.*[1]

1. Robert C. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design* (Boston, MA: Pearson Education, 2018), p. 58.

SOLID principles: not just for OOP



[2]

2. Robert C. Martin, August 2019, https://twitter.com/unclebobmartin.

The SOLID principles, listed

- **S** The Single-Responsibility Principle.
- **O** The Open–Closed Principle.
- **L** The Liskov Substitution Principle.
- **I** The Interface-Segregation Principle.
- **D** The Dependency-Inversion Principle.

The Single-Responsibility Principle, defined

A module should have one, and only one, reason to change.[3]

A module should be responsible to one, and only one, user or stakeholder.[4]

A module should be responsible to one, and only one, actor.[5]

3. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design*, p. 62.
4. Martin, p. 62.
5. Martin, p. 62.

The Single-Responsibility Principle, continued

The *actor* or *user/stakeholder* idea means code that accounting asks you to change shouldn't affect code that SEO asks you to change, for example.

Let's look at code.

The Open–Closed Principle, defined

A software artifact should be open for extension but closed for modification.[6]

Let's look at some old code in Baerlauch.

6. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design*, p. 70.

The Liskov Substitution Principle, defined

Let $\phi(x)$ be a property provable about objects x of type T. Then $\phi(y)$ should be true for objects y of type S where S is a subtype of T.[7]

7. Barbara H. Liskov and Jeannette M. Wing, "A behavioral notion of subtyping," *ACM Transactions on Programming Languages and Systems* 16 (1994): p. 1812.

The Liskov Substitution Principle, continued

Put in plainer language, this means that we mustn't subtype something in a way that means the subtype can't be used everywhere we could use the supertype.

The Liskov Substitution Principle, continued

Liskov seems to apply to functional code in terms of generics, but because TypeScript enforces structural types, the type system should generally enforce correct subtyping.

The Interface-Segregation Principle, defined

Keep interfaces small so that users don't end up depending on things they don't need.[8]

8. Robert C. Martin, "Solid Relevance," October 2020, https://blog.cleancoder.com/uncle-bob/2020/10/18/Solid-Relevance.html.

The Dependency-Inversion Principle, defined

Depend in the direction of abstraction. High level modules should not depend upon low level details.[9]

9. Martin, "Solid Relevance."

Bibliography


Liskov, Barbara H., and Jeannette M. Wing. “A behavioral notion of subtyping.” *ACM Transactions on Programming Languages and Systems* 16 (1994): 1811–1841.


Martin, Robert C., August 2019. https://twitter.com/unclebobmartin.


———. *Clean Architecture: A Craftsman’s Guide to Software Structure and Design*. Boston, MA: Pearson Education, 2018.


———. “Solid Relevance,” October 2020. https://blog.cleancoder.com/uncle-bob/2020/10/18/Solid-Relevance.html.