**S Single Responsibility**, SRP  
**Принцип единой ответственности**

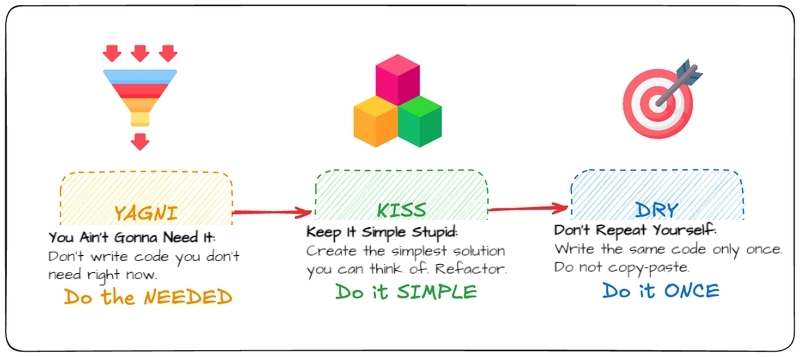
Компонент (модуль, класс, функция, хук) должен иметь одну и только одну причину для изменения, то есть отвечать за одну задачу или функциональность.  
Прямо следует из принципов KISS и DRY. Нужен для упрощения тестирования, отладки и повторного использования кода.

***В React:****Компонент должен рендерить UI, управлять своим собственным состоянием (если это необходимо для рендеринга) и больше ничего. Логику данных (API-вызовы), навигацию, бизнес-правила лучше выносить в отдельные модули, хуки или контекст.*

**O Open–Closed**, OCP  
**Принцип открытости/закрытости**

Сущности (компоненты, модули, функции) должны быть открыты для расширения (через наследование, композицию, HOC) но закрыты для модификации.  
В основе лежат принципы ООП: **Инкапсуляция** (сокрытие деталей реализации) и **Полиморфизм** (возможность использовать разные реализации через единый интерфейс). В React композиция предпочтительнее наследования.

***В React:****Используйте Композицию Компонентов (children, render props), Высокоуровневые Компоненты (HOC) или хуки для добавления новой функциональности, не ломая и не изменяя исходный код существующего компонента.*

**L Liskov Substitution**, LSP  
**Принцип подстановки Лисков**

Объекты (компоненты, классы) должны быть заменяемыми на экземпляры их подтипов (потомков), не нарушая работу программы. Поведение наследника должно быть ожидаемым и совместимым с поведением родителя.

Является строгой формулировкой принципа **Наследования** в ООП. Наследование должно дополнять, а не замещать или нарушать контракт базового класса.

***В React:****Если вы создаете компонент-обертку (Button -> IconButton), он должен принимать и корректно обрабатывать все пропсы базового компонента, а не ломать его стили или логику.*

**I Interface Segregation**, ISP  
**Принцип разделения интерфейсов**

Клиенты (компоненты, модули) не должны зависеть от интерфейсов (пропсов, методов), которые они не используют. Создавайте узкоспециализированные интерфейсы.  
Связан с принципами KISS и DRY, а также с **Абстракцией** в ООП. Уменьшает связанность и предотвращает "захламление" пропсами.

***В React:****Вместо одного большого компонента с десятками пропсов (onClick, onChange, onSubmit...), разбейте его на несколько мелких. Или используйте деструктуризацию и пропс ...rest, чтобы передавать только нужные атрибуты (например, в input).*

**D Dependency Inversion**, DIP  
**Принцип инверсии зависимостей**

Мы должны полагаться на абстракции, а не на конкретные реализации. Компоненты ПО должны иметь низкую связность и высокую согласованность.  
Заботиться нужно не о том, как что-то устроено, а о том, как оно работает.  
Это основа **Абстракции** в ООП. Позволяет управлять зависимостями извне, что критически важно для тестирования (моки, заглушки) и гибкости архитектуры.

*Представь, что твой компонент — это кофемашина. Ей нужен только «абстрактный» кофе: «добавить\_кофе()». Как именно зёрна обжарены, молоты, заварены — ей всё равно.  
Ты подсовываешь ей разные капсулы (мок, реальные зёрна, тестовая заглушка) — и она работает без доработки.*