Dưới đây là 20 nguyên tắc và định luật thông dụng trong lập trình giúp viết mã dễ bảo trì, mở rộng, và ít lỗi:

KISS (Keep It Simple, Stupid): Giữ cho mã nguồn đơn giản, dễ hiểu và tránh phức tạp hóa.

DRY (Don't Repeat Yourself): Tránh lặp lại mã nguồn. Thay vào đó, tái sử dụng code qua các hàm, module hoặc class.

SOLID Principles:

S: Single Responsibility Principle - Mỗi class nên chỉ có một lý do để thay đổi.

O: Open/Closed Principle - Code nên mở rộng được nhưng không thay đổi trực tiếp.

L: Liskov Substitution Principle - Subclass có thể thay thế cho superclass mà không ảnh hưởng đến tính đúng đắn của chương trình.

I: Interface Segregation Principle - Tạo nhiều interface cụ thể thay vì một interface lớn.

D: Dependency Inversion Principle - Mô-đun cấp cao không nên phụ thuộc vào mô-đun cấp thấp.

YAGNI (You Ain't Gonna Need It): Không phát triển tính năng mà bạn nghĩ rằng "có thể cần" trong tương lai; chỉ phát triển tính năng khi nó thực sự cần thiết.

Law of Demeter: Giảm số lượng phụ thuộc giữa các module và chỉ nên tương tác với các thành phần “gần gũi” (ít phụ thuộc đến các module khác).

Murphy's Law: Nếu có thể có lỗi, chắc chắn sẽ có lỗi. Luôn kiểm tra các tình huống có thể phát sinh lỗi và xử lý chúng.

Boy Scout Rule: "Hãy để code sạch hơn khi bạn rời đi." Luôn cải tiến code mà bạn chỉnh sửa hoặc tiếp xúc.

Fail Fast: Phát hiện lỗi sớm nhất có thể để dễ dàng xác định và sửa chữa. Giúp phát hiện lỗi ngay từ giai đoạn đầu, thay vì khi đã triển khai.

Composition over Inheritance: Sử dụng thành phần (composition) thay vì kế thừa để tăng tính linh hoạt và tránh những rắc rối do kế thừa sâu gây ra.

Separation of Concerns: Mỗi module chỉ nên thực hiện một nhiệm vụ chính để dễ quản lý, bảo trì, và test.

Refactoring: Luôn cải tiến mã (mà không thay đổi chức năng) để giữ mã sạch, dễ hiểu và dễ bảo trì.

Encapsulation: Giữ dữ liệu và hành vi liên quan trong một thực thể riêng biệt, tránh để lộ chi tiết nội bộ không cần thiết.

TDD (Test-Driven Development): Viết test trước khi viết code để đảm bảo mã hoạt động như mong đợi và dễ phát hiện lỗi.

The Principle of Least Astonishment: Thiết kế code theo cách người dùng (hoặc lập trình viên khác) sẽ không ngạc nhiên khi đọc hoặc sử dụng.

Pareto Principle (80/20 Rule): 80% kết quả đến từ 20% công sức. Đầu tư thời gian cho các phần code có nhiều tác động nhất.

Optimize Last: Không tối ưu hóa quá sớm. Tập trung vào viết code chính xác và rõ ràng trước, tối ưu hóa sau.

Inversion of Control (IoC): Trao quyền kiểm soát cho bên ngoài (dùng Dependency Injection) để giảm phụ thuộc và tăng tính linh hoạt.

Leaky Abstraction: Các lớp trừu tượng nên che giấu chi tiết phức tạp. Đừng để lộ chi tiết thực thi ra ngoài các lớp trừu tượng.

Big O Notation: Hiểu về độ phức tạp của thuật toán để tối ưu hóa hiệu suất, đặc biệt là với các chương trình xử lý nhiều dữ liệu.

Single Source of Truth (SSOT): Giữ một nguồn thông tin chính xác duy nhất cho một phần dữ liệu để giảm rủi ro mâu thuẫn dữ liệu.