perplexity

What are Code Smells?

A **code smell** is a characteristic in source code that indicates a deeper problem, even if the code works technically correct. It is not a bug (the code runs without crashing), but rather a sign of poor design that makes the software hard to maintain, extend, or test.[1] [2]

Think of it like a messy kitchen: you can still cook dinner (the code runs), but finding the right tools is hard, and you are more likely to make mistakes or start a fire (bugs) later.[3]

Common Examples of Code Smells:

- **Long Method:** A function that contains hundreds of lines of code and does too many things.
- **God Class:** A single class that controls almost everything in the system.
- **Duplicated Code:** Copy-pasting the same logic in multiple places (Dry Principle violation).[4]
- **Shotgun Surgery:** When making a small change (e.g., adding a database column) forces you to edit 10 different classes at once.[1]

Best Practices for Clean Code

To avoid code smells and keep your application healthy, follow these industry-standard practices:

1. SOLID Principles

- **Single Responsibility Principle (SRP):** A class or method should do *one* thing well. If you have a `User` class that handles *login*, *database saving*, and *email sending*, break it into three separate classes.[1]
- **Open/Closed Principle:** Code should be open for extension but closed for modification. You should be able to add new features (like a new "Shipping Strategy") without rewriting existing, tested code.

2. KISS (Keep It Simple, Stupid)

- Avoid over-engineering. If a simple `if-else` works, don't build a complex "Rule Engine" unless you absolutely need it. Complexity breeds bugs.[5]

3. DRY (Don't Repeat Yourself)

- Extract common logic into utility functions or shared services. If you fix a bug in one place, it should be fixed everywhere. Copy-pasted code means you have to fix the same bug 5 times.[4]

4. Meaningful Naming

- Variables and functions should explain *what* they do.
  - **Bad:** `int d; // elapsed time in days`
  - **Good:** `int elapsedTimeInDays;`
- Code is read 10x more often than it is written. Write it for humans, not just computers.[6]

5. Code Reviews & Refactoring

- **Refactor Early:** Don't wait for the code to become unmanageable. If you see a "smell," clean it up immediately (Boy Scout Rule: "Leave the campground cleaner than you found it").[7]
- **Peer Reviews:** Have another developer read your code. They will spot "smells" you missed because you were too focused on just making it work.

⁂

1. https://en.wikipedia.org/wiki/Code_smell
2. https://opsera.ai/blog/what-is-code-smell/
3. https://www.jetbrains.com/pages/static-code-analysis-guide/code-smells/
4. https://www.sonarsource.com/resources/library/code-smells/
5. https://dev.to/bytebodger/what-is-a-code-smell-38i1
6. https://www.legitsecurity.com/aspm-knowledge-base/code-smells
7. https://www.techtarget.com/searchsoftwarequality/tip/Understanding-code-smells-and-how-refactoring-can-help
8. https://refactoring.guru/refactoring/smells
9. https://www.geeksforgeeks.org/blogs/code-smell-a-general-introduction-and-its-type/
10. https://jellyfish.co/library/code-smells/