





# CODE SMELLS



# CONTENTS

1. What is code smell
2. How to fix it





# INTRODUCTION TO CODE SMELLS

Software development guru **Martin Fowler** credits his collaborator **Kent Beck** for coining the term “code smell”.

Fowler defines it as such:

“**A CODE SMELL IS A SURFACE INDICATION THAT USUALLY CORRESPONDS TO A DEEPER PROBLEM IN THE SYSTEM.**



Fowler highlights two important nuances packed into the term “smell.”

First, you detect smells effortlessly. The same is true of code smells: they jump out to you immediately.



Second, Fowler notes that code smells signify underlying problems. **They suggest, rather than confirm, that bad code exists.** Here's an analogy: If you get a whiff of that stinky feet smell, you could be smelling stinky feet.

**Or you could be smelling the parmesan cheese.**





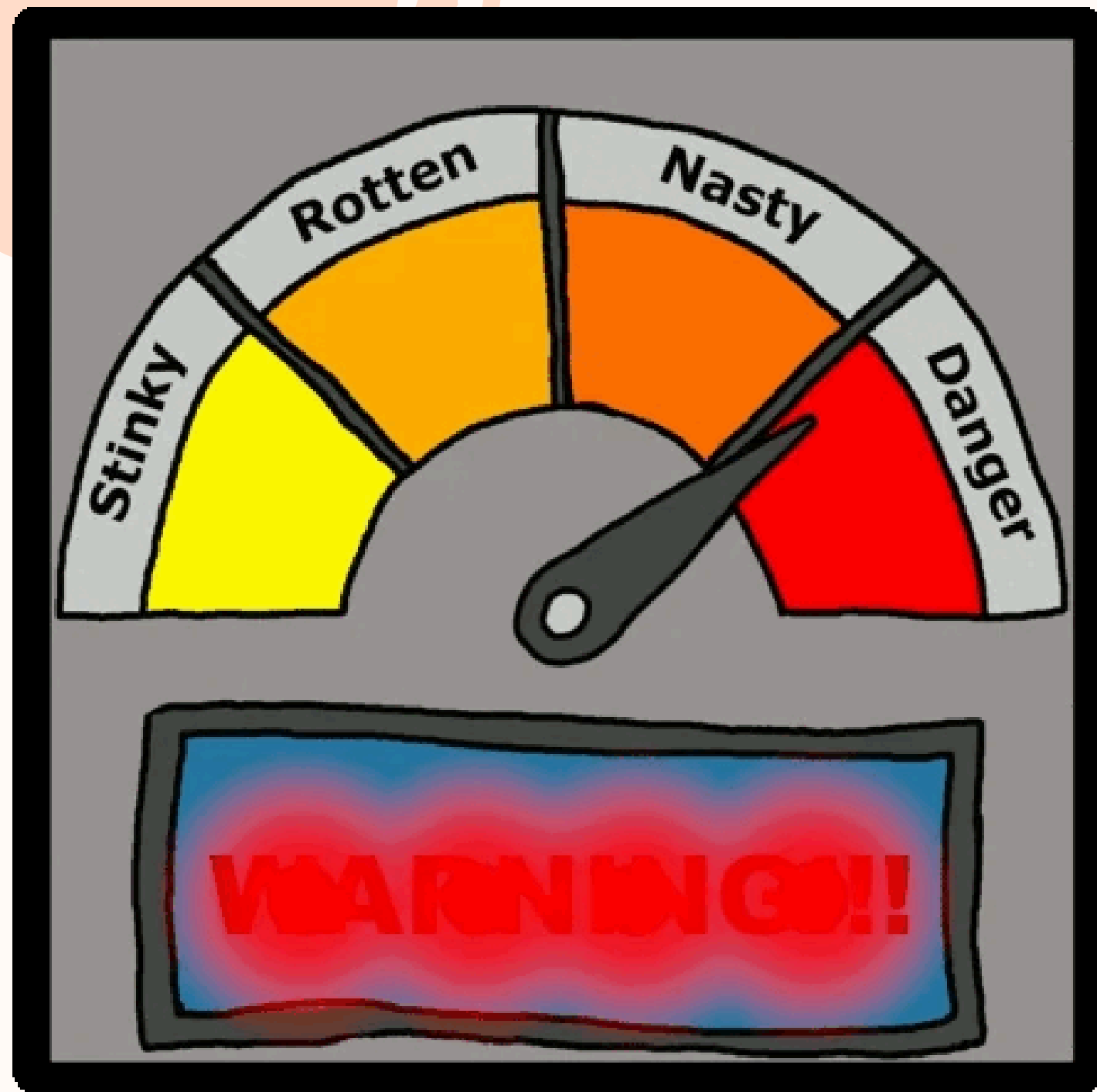
# WHAT ARE CODE SMELLS?

They are indicators in the code that suggest deeper issues within the application or codebase.

They aren't bugs or errors but are visible violations of fundamental design principles that can lead to poor code quality and technical debt. Even though the code will still compile and run, it may hinder the development process and expose the program to performance and security problems in the future.

Essentially, code smells highlight underlying issues that contribute to technical debt.

# SMELLY CODE CAN BECOME ROTTEN CODE



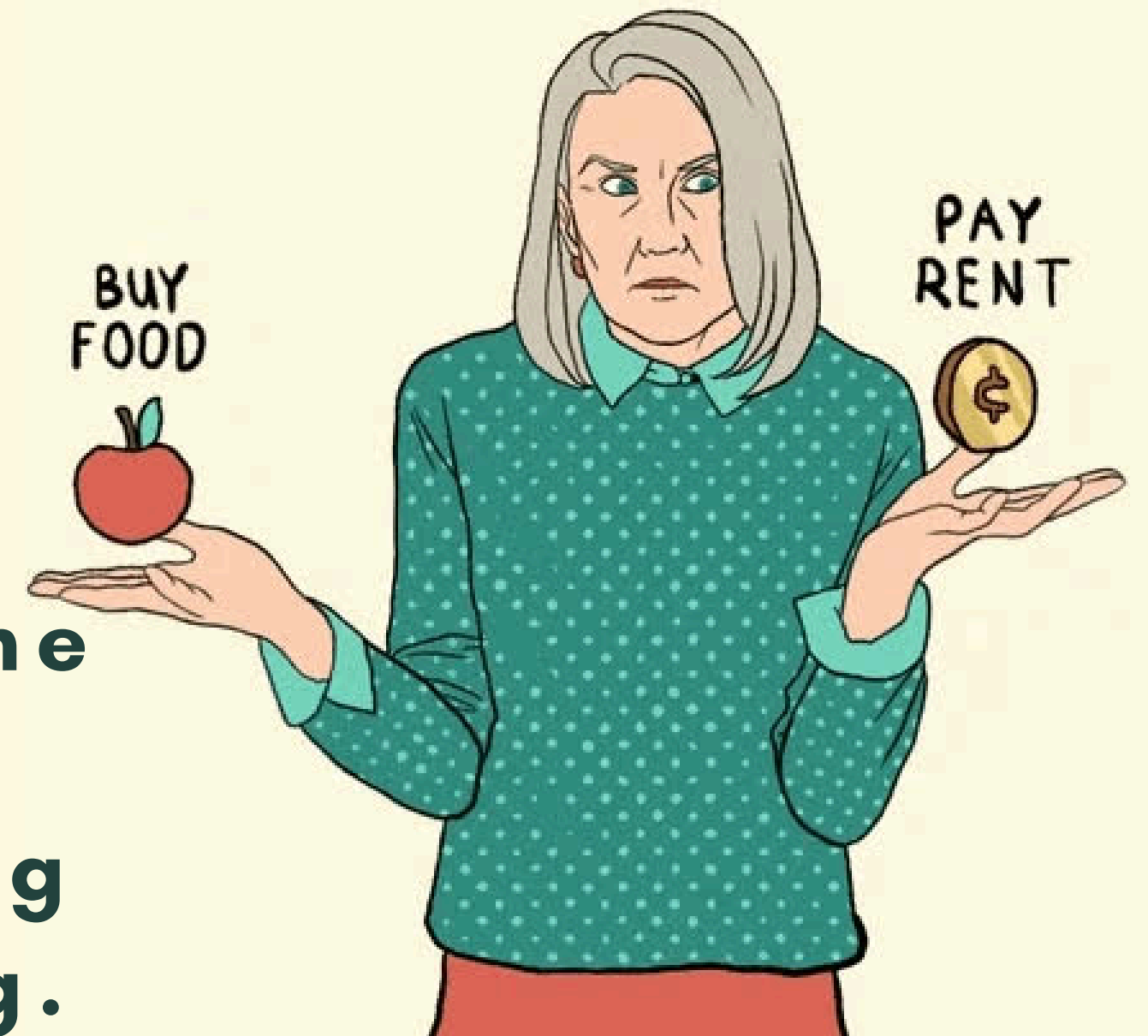
Code that smells can degrade over time, becoming rotten. This degradation process, known as code rot, is often fueled by technical debt. While some technical debt is inevitable, it can accumulate quickly and significantly slow down development. An example of this is a class that takes on too many responsibilities, turning into a "god class."



**Technical debt**, also called tech debt or code debt, arises when development teams prioritize fast delivery over perfect code.

This means cutting corners to meet deadlines, which later requires refactoring.

Like financial debt, technical debt compounds over time: **as engineers write more code to work around the smelly code, the degradation spreads, making future refactoring more difficult and time-consuming.**



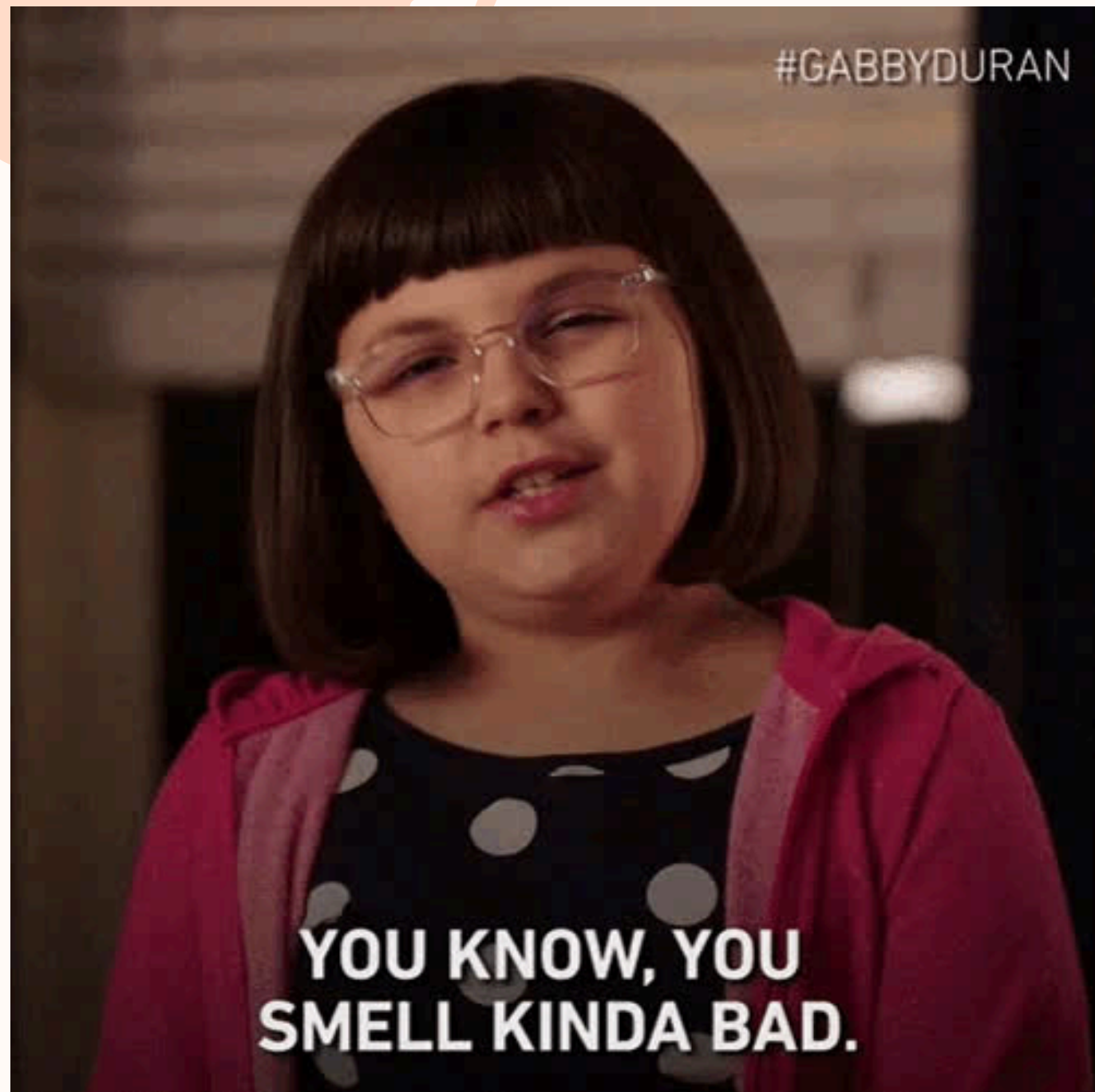
This underscores the **importance of eliminating code smells**. It's not just about making code look good; it's about preventing technical debt and code rot, keeping development sustainable and efficient.

MUST. SNIFF. EVERYTHING.



When teams can develop quickly, shipping new features and adding value, they are more satisfied and productive.

# COMMON TYPES OF CODE SMELLS



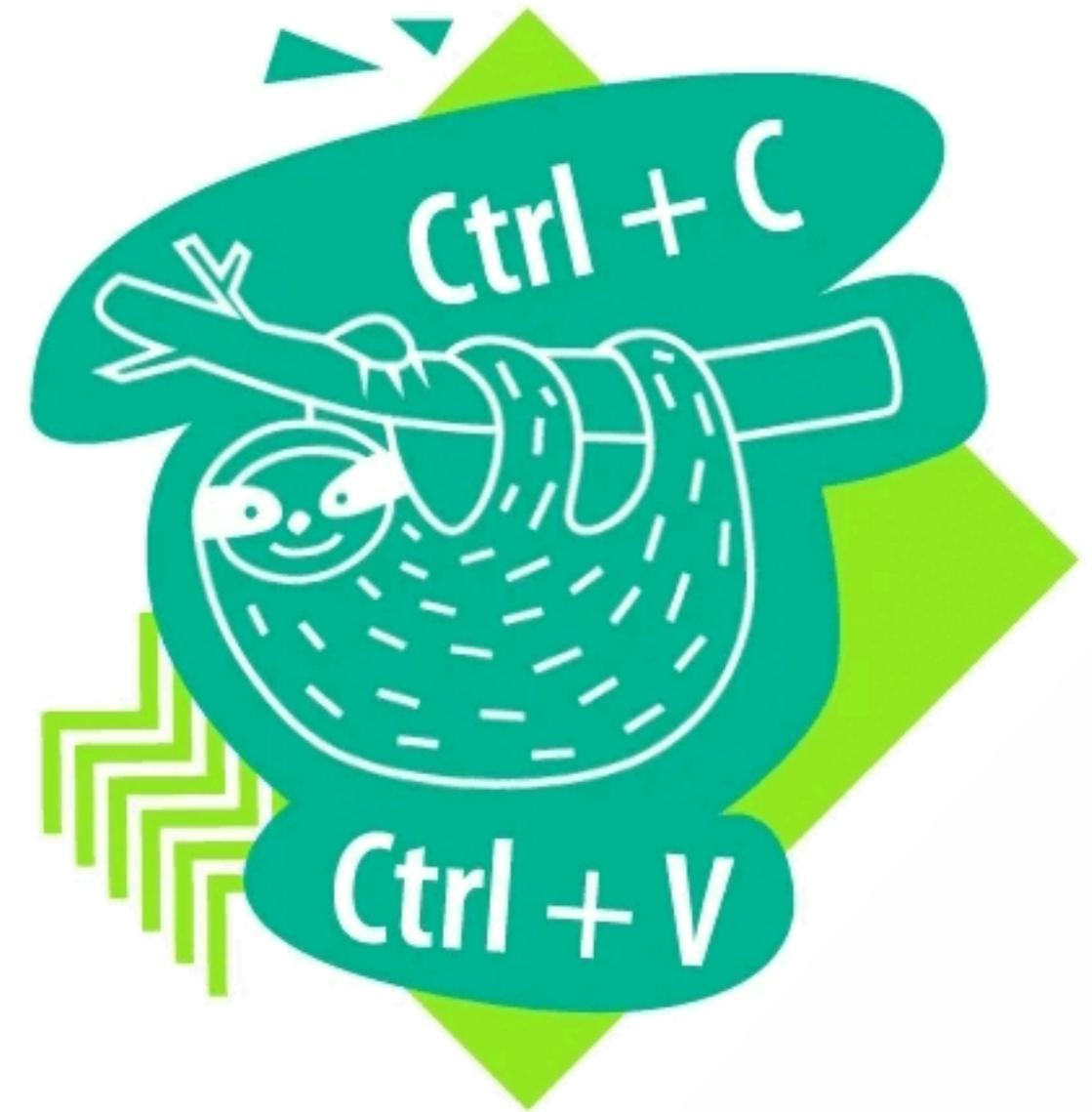
Typically, code smell occurs when developers fail to write code in accordance with design principles and standards.

Here are the most common code smells to help you get a head start on improving your code quality:

## DUPLICATE CODE

Identical or similar code in multiple places complicates updates.

Extract duplicates into a single method.





# DEAD CODE

**THAT STINKS**



# IMPROPER NAMES

A dead code is a section of code in the source code of a program that is of no use and is never reached or called. Use tools to identify and remove it.

Improper naming of variables, classes, and functions indicates that your code is not clean. This makes code hard to read, understand, and maintain. Use descriptive names for clarity.

## MIDDLE MAN

Middle man refers to the class that delegates work to another class and doesn't have any independent functionality. Fix this code smell by removing the middle man.



# LONG PARAMETER LIST

A long list of parameters in a method or class is a code smell. This hinders code readability and reusability and makes it prone to bugs and errors. More than three or four parameters are not advisable.

This code smell can be fixed by introducing a parameter object.

Create a new object from the list of parameters that match and transfer it as a single argument.



## DATA CLUMPS

Groups of data appearing together repeatedly. This makes it hard to centrally control their behavior. Refactor into objects.

## LONG FUNCTIONS

Overly long functions handle too much. Break them into smaller, task-specific functions.

## PRIMITIVE OBSESSION

Overuse of primitive data types. Replace with small classes or objects.





# FEATURE ENVY

Feature envy occurs when a class with a method is overly dependent on another class.

Here are the steps to fix a feature envy:

- If a function invokes methods on another object, use Move Function to move the function to the data.
- If only a part of a function is envy, first use Extract Function and then Move Function.
- In case, if the function envies several modules, use Extract Function to break the function into several pieces, and then the Move Function.

# COMMENTS


Code comments are a code smell. Comments in your code are definitely a good practice, but leaving the **commented-out code** in place is a code smell.

Comments can be helpful to explain the logic, but overusing comments for every step of the program creates excessive noise in your code and hampers readability. Your code should be **self-explanatory** and use comments as minimally as possible.



# DEALING WITH CODE SMELLS



- **Refactoring:** Regularly restructure code to improve readability and maintainability without altering functionality. Tools like SonarQube and Eclipse IDE can help automate this process.
  - **Continuous Integration:** Implement CI/CD (Continuous Deployment) to track and test incremental code changes, reducing the risk of code smells.
  - **Code Review:** Regularly review code with peers to identify and address code smells early. Incorporate feedback loops to continuously improve code quality.
- 



# QUESTIONS?



# THANKS A LOT!





## SOURCES

<https://refactoring.guru/refactoring/smells>

<https://www.sonarsource.com/learn/code-smells/>