UFMG

lab-soft
software engineering laboratory

Code Smells

Saymon Souza
MSc Student in Computer Science | silvasouzasaymon@gmail.com

Definition

- A code smell is any symptom in a program's source code that ***may indicate a potential problem.***
    - Determining what is or is not a code smell is subjective;
    - It varies depending on the programming language, the developers, and the development methodologies.

Why you should care about code smells?

- Detecting and fixing code smells is crucial for maintaining a healthy, scalable, and maintainable codebase.
    - Reduces Complexity;
    - Eases Future Changes;
    - Fewer Bugs.

Some code smells

Refused Bequest	**Large Class**	**Long Method**	Comments
Divergent Change	**Shotgun Surgery**	**Feature Envy**	**Long Parameter List**
Primitive Obsession	Switch Statements	Lazy Class	Speculative Generality
Temporary Field	Message Chains	Middle Man	**Intensive Coupling**
Dispersed Coupling	Data Clumps	**Data Class**	Incomplete Library Class
Parallel Inheritance Hierarchies		Alternative Classes with Different Interfaces	

Long Method

- Long Method is a code smell ***where a method is excessively long*** and ***does too many things***, making it hard to understand, test, and maintain. Key characteristics:
    - Scrollbar required to read the entire method.
    - Multiple levels of nesting (deep if/for/try blocks).
    - Comments explaining sections (a hint that chunks could be separate methods).

Long Method - Example

- https://github.com/sssouza/code-smell-examples/blob/main/long_method.java

Long Parameter List

- Long Parameter List is a code smell where a ***method or constructor has too many parameters***, making it hard to understand, test, and maintain.
    - Related parameters should be grouped into an object;
    - Frequently adding new parameters to existing methods;
    - Parameters are often passed together in multiple methods.

Long Parameter List - Example

- https://github.com/sssouza/code-smell-examples/blob/main/long_parameter_list.java

Large Class

- Large Class is a code smell where ***a class has too many responsibilities, methods, or fields***, making it difficult to understand, test, and maintain.

    - Multiple unrelated concerns (e.g., a User class handling authentication, profile management, and billing).

Large Class - Example

- https://github.com/sssouza/code-smell-examples/blob/main/large_class.java

Refused Bequest

- Refused Bequest is a code smell where a ***subclass inherits from a parent class but does not use (or actively rejects)*** some of the inherited methods or properties. Key characteristics:
    - Subclass overrides a parent method to do nothing (or throw an exception).
    - Subclass ignores inherited fields/methods.
    - Subclass renames or repurposes inherited methods (breaking polymorphism).

Refused Bequest - Example

- https://github.com/sssouza/code-smell-examples/blob/main/refused_bequest.java

Feature Envy

- Feature Envy is a code smell where a ***method in one class excessively uses methods or data from another class***, suggesting it might belong elsewhere. Key characteristics:
    - A method makes more calls to another class than its own.
    - Violates encapsulation by pulling data out of another object to process it.

Feature Envy - Example

- https://github.com/sssouza/code-smell-examples/blob/main/feature_envy.java

Shotgun Surgery

- Shotgun Surgery is a code smell where a ***single change requires many small modifications across multiple classes/files***, indicating that responsibilities are poorly organized. Key problems:

  - Fragile Code: A simple change (e.g., adding a new field) forces edits in many places.
  - High Maintenance: Hard to track all affected files.
  - Bug-Prone: Easy to miss a required change.

Shotgun Surgery - Example

- https://github.com/sssouza/code-smell-examples/blob/main/shotgun_surgery.java

Intensive Coupling

- Intensive Coupling is a code smell where a class or module is ***overly dependent on the internal details of another class/module***, making the system rigid and hard to modify.

    - Unlike dispersed coupling (which spreads dependencies across many classes), intensive coupling focuses on deep, inflexible relationships between specific components.

Intensive Coupling - Example

- https://github.com/sssouza/code-smell-examples/blob/main/intensive_coupling.java

Dispersed Coupling

- Dispersed Coupling is a code smell that occurs when a module or class ***depends on multiple other modules or classes in a scattered or decentralized way***, leading to a lack of cohesion and maintainability issues.

    - Multiple Dependencies: A single module relies on many different modules (directly or indirectly), making it difficult to understand or modify.

Dispersed Coupling - Example

- https://github.com/sssouza/code-smell-examples/blob/main/dispersed_coupling.java

Data Class

- Data Class is a code smell ***where a class only holds data (fields + getters/setters) without meaningful behavior,*** essentially acting as a "dumb container". Key problems:

    - Anemic Domain Model: Business logic gets scattered outside the class (often in "manager" or "service" classes).
    - Poor Encapsulation: External code manipulates the class's data directly.

Data Class - Example

- https://github.com/sssouza/code-smell-examples/blob/main/data_class.java

Experiment

How will the experiment work?

- Each student **MUST** answer two forms:
    - Consent: Standard consent form where participants are giving informed consent to participate in the experiment. Here's the link!
    - Background: This form collects information about your professional experience and knowledge of Java and code smells. Here's the link!

How will the experiment work?

- Each student has a tab in the allocation table. Look for your @aluno.ufop.edu.br e-mail handle there.
    - In your tab you will find 10 codes and their respective forms.
        - Also, you will see a DONE checkbox to mark your progress.
        - Codes were assigned randomly, so you might evaluate more codes of one type instead of one of each type of smell.
- **Be careful while answering to make sure you are answering the right form for each code!**

Finishing the experiment and grading

- The experiment ends when all 10 codes are evaluated in their respective forms and the two previous forms (consent and background) are answered.
- Grades will be split in **two** parts:
  - Completion: Did you made all 10 evaluations and answered both background forms?
  - Quality of answers: Did you try to give good evaluations?

Tips!

- Take a look at the code smells definitions and examples in this presentation whenever you need!
    - Detecting code smells is hard ***even*** for LLMs and Tools.

- There is no right or wrong answer, but **there is good and bad justifications**!

- **If you have any questions, ask me immediately!**

UFMG

lab-soft
software engineering laboratory

Code Smells

Saymon Souza
MSc Student in Computer Science | silvasouzasaymon@gmail.com