Introduction

CODESAI

Refactoring

Process of changing the code to make it easier to understand and cheaper to modify but preserving its observable behavior.

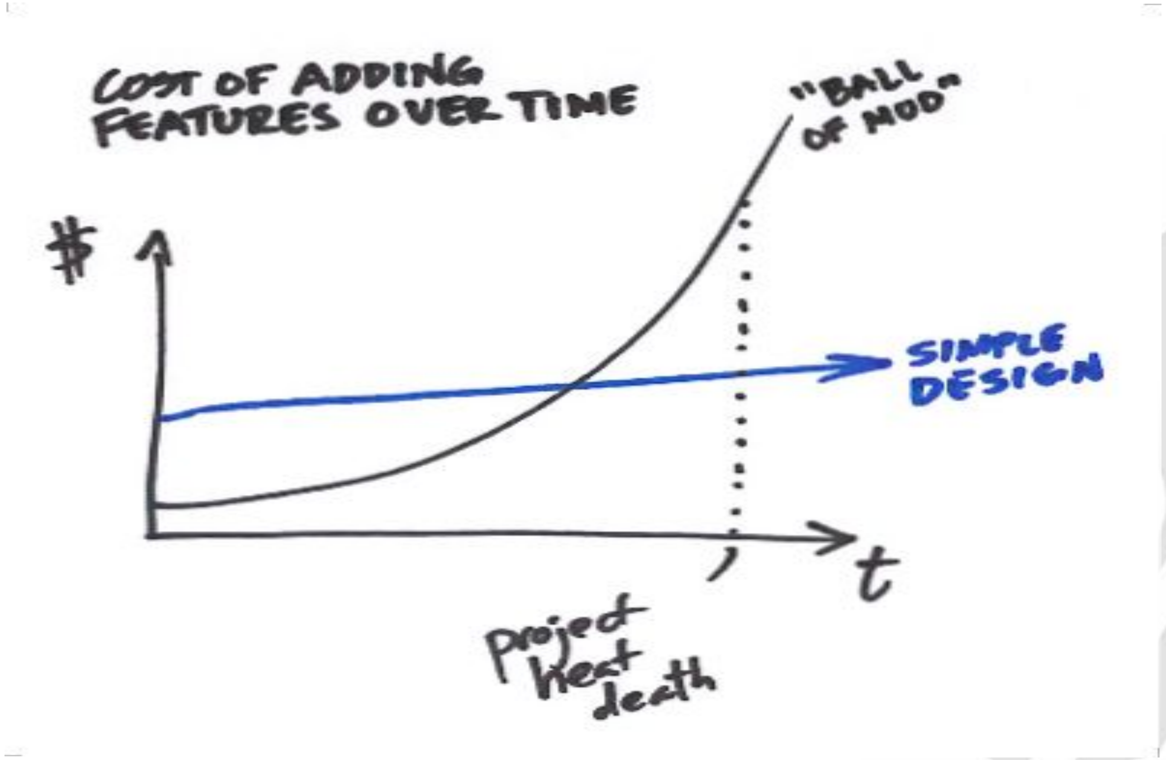
What Refactoring is not

- Refactoring does not include just any changes in a system.
- Refactoring is not rewriting from scratch.
- Refactoring is not just any restructuring intended to improve.

Why Refactoring?

- It makes sense economically.
- It helps us ship more value faster.

Cost of adding features



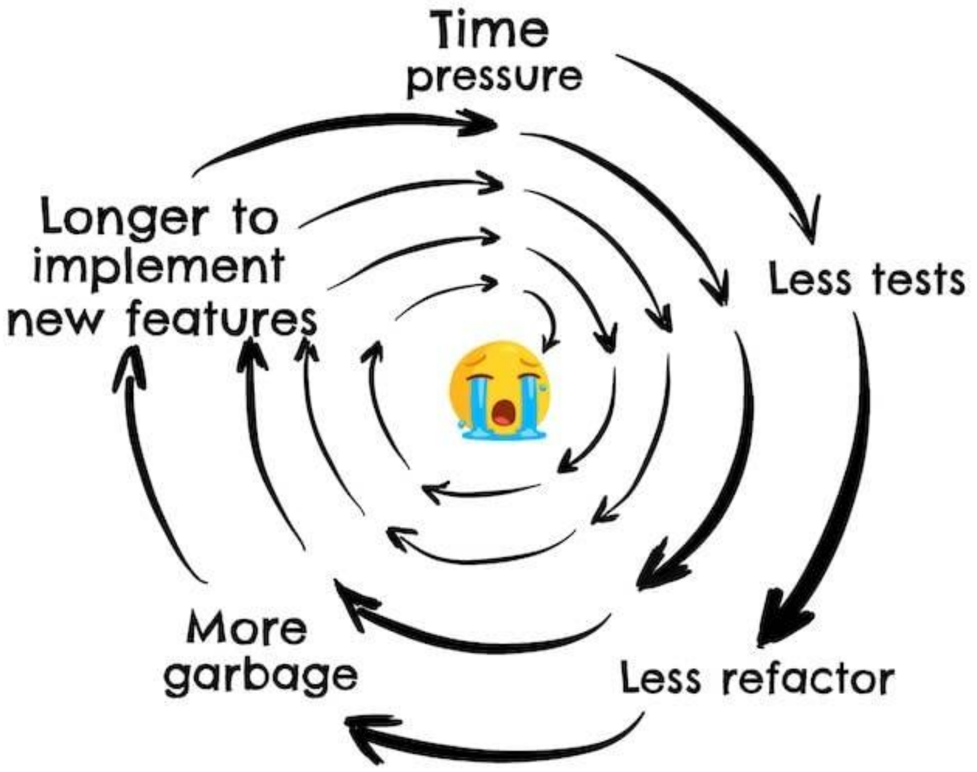
Costs of poor internal quality

- The lack of internal quality carries an associated cost.
  - How easy is to **find** the code?
  - How easy is to **change** this code?
  - How easy is to **test** my changes?

Costs of poor internal quality

- The lack of internal quality carries an associated cost.
  - How easy is to **find** the code?
  - How easy is to **change** this code?
  - How easy is to **test** my changes?
- There is a correlation between internal quality and productivity.

Effects on Team Morale



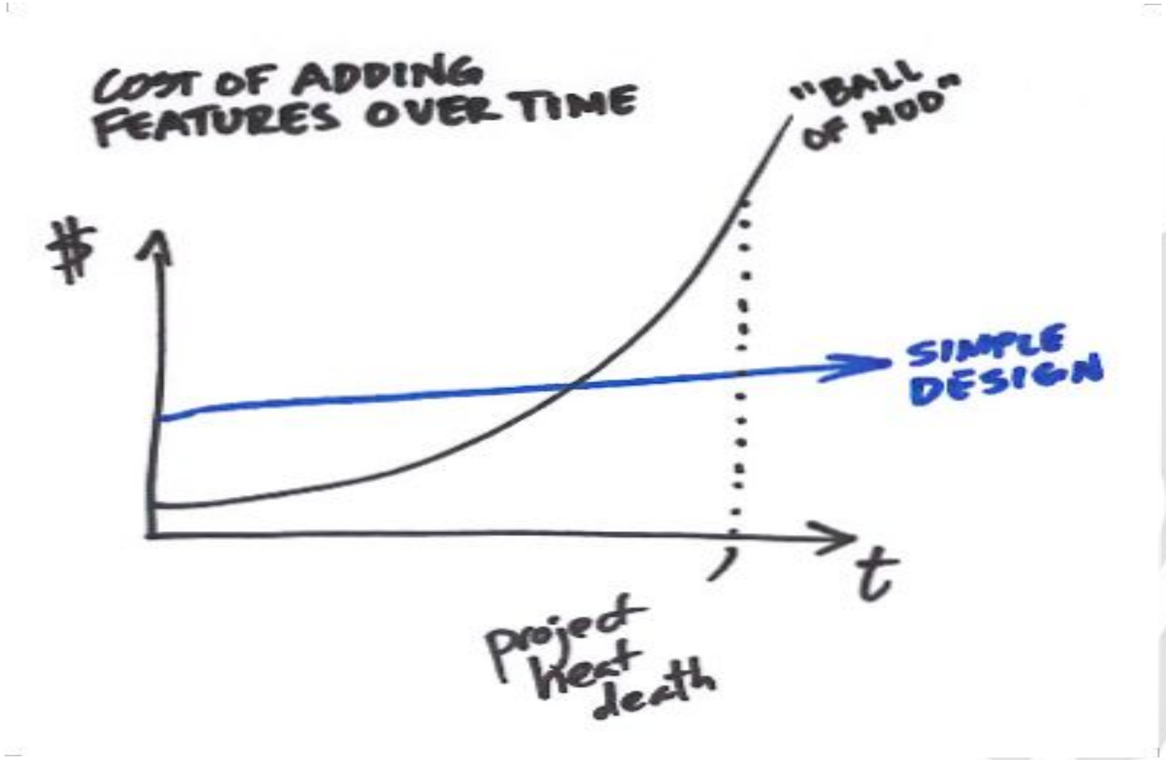
How can we keep code from decaying?

Constant refactoring keeps **evolvability**

Evolvable

- It’s easy to **find** the code you have to change
- You can **understand** it
- It’s easy to **change**
- It’s easy to **test** the changes

Sustainability



Habitable Code

“Habitability enables programmers, coders, bug-fixers, and people coming to the code later in its life to understand its construction and intentions and to change it comfortably and confidently.”

Richard Gabriel

Precondition -> tests

- Refactoring preserves **observable behavior**.
- If you want to refactor, the essential precondition is having **solid tests**.
- It scales better if testing is automated.

Emergent Design

- Refactoring lowers the cost of change.
- This changes the balance point between up front and emergent design.
- We still need some design up front but not a lot, (see Is Design Dead?)

Ok, refactoring is great, but…

- How do we know what to refactor?
- How can we recognize design problems?

Code Smells

- Warning signs about potential problems or flaws in the code.
- A sign, not a guarantee.
- Some are obvious, some are not. Some mask other problems.
- They usually describe localized problems.

Code Smells & Refactorings

For each code smell, there are related refactorings that might be used to remove it.

Many code smells...

Duplicate code

Long method

Large class

Long parameter list

Primitive obsession

Data clumps

Switch statements

Shotgun surgery

Divergent Change

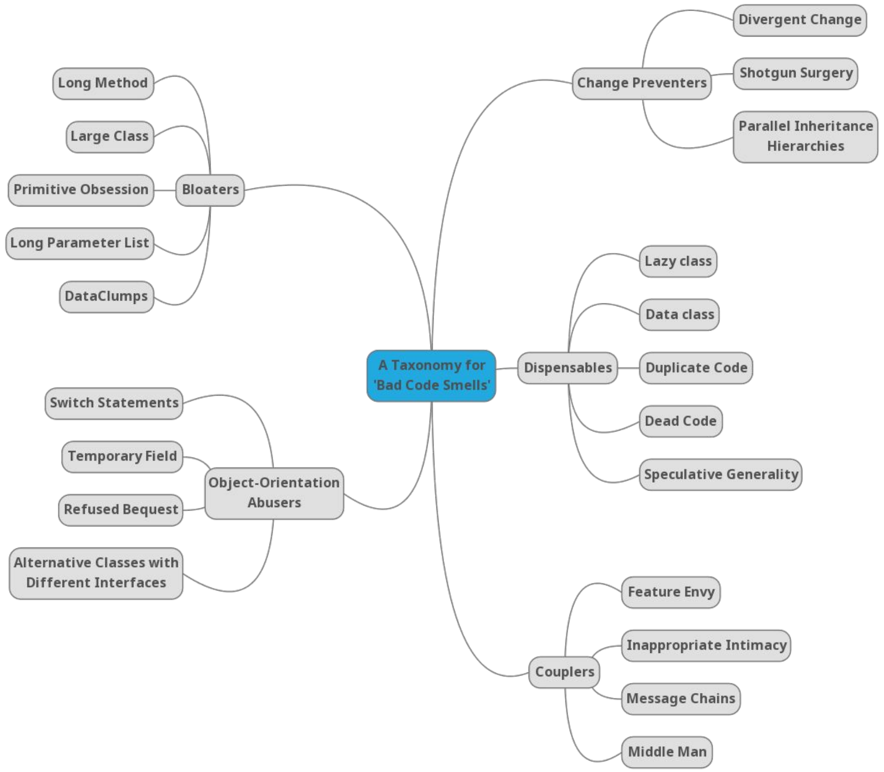
Comments (deodorant)

Boolean parameter

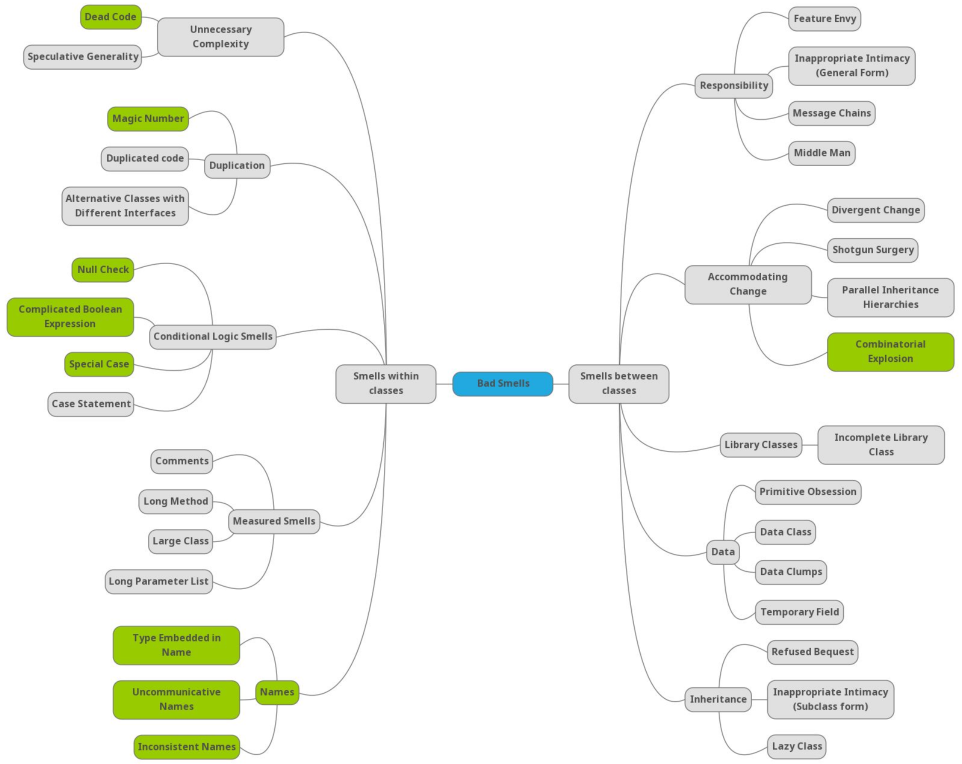
Message chains

How do we organize all this knowledge?

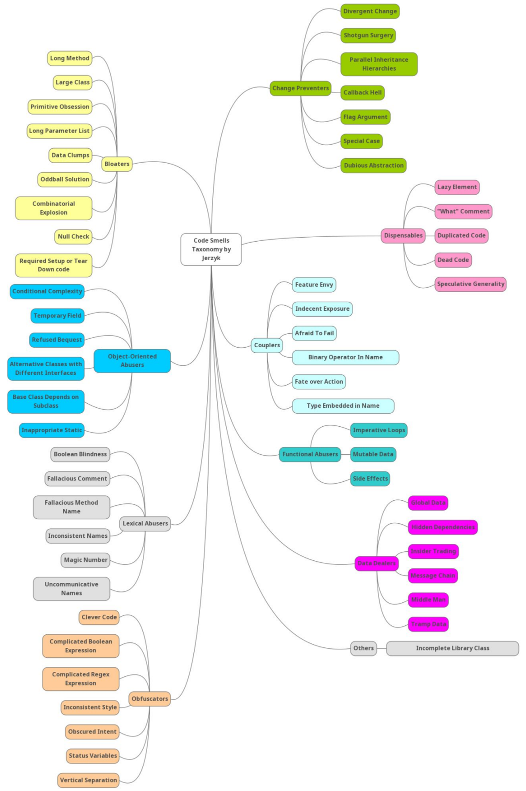
Mäntylä & Lassenius’ classification



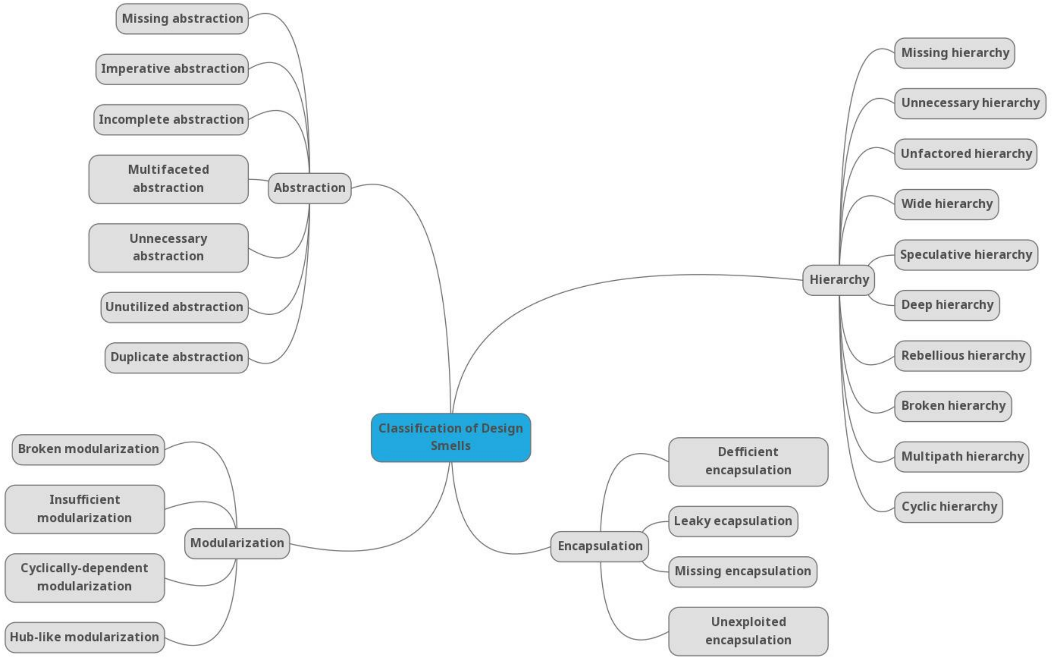
Wake's classification



Jerzyc’s classification



Surynarayana, Samarthyam and Sharma's classification


Classification of Design Smells
Abstraction
Missing abstraction
Imperative abstraction
Incomplete abstraction
Multifaceted abstraction
Unnecessary abstraction
Unutilized abstraction
Duplicate abstraction
Hierarchy
Missing hierarchy
Unnecessary hierarchy
Unfactored hierarchy
Wide hierarchy
Speculative hierarchy
Deep hierarchy
Rebellious hierarchy
Broken hierarchy
Multipath hierarchy
Cyclic hierarchy
Modularization
Broken modularization
Insufficient modularization
Cyclically-dependent modularization
Hub-like modularization
Encapsulation
Defficient encapsulation
Leaky ecapsulation
Missing encapsulation
Unexploited encapsulation