Things I Wish I Knew Before Becoming Software Engineer

Clean Code and Code Smell

A talk by @hezbymuhammad

About Me

Hezby

Lead Software Engineer
at Mekari

https://twitter.com/muhammadhezby
http://github.com/hezbymuhammad

Agenda

1. Introduction
2. What is clean code?
3. Why do we need Clean Code?
4. Code Smell
5. Refactoring pattern for each Code Smell
6. QnA

Introduction

Software Engineering at School

- Isolated problem (i.e. sorting array)
- Well defined solution (i.e. binary tree)
- Known time to solve (i.e. assignment for 3 days)

Software Engineering at Work

- Unknown root cause
- Unknown solution
- Unknown time

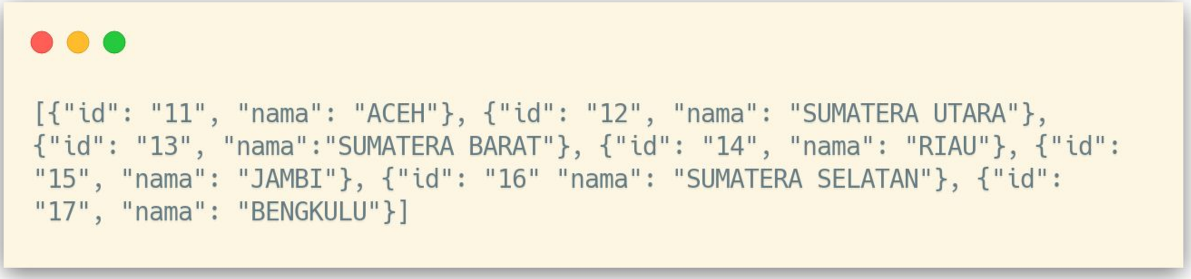
Software Engineering at Work

Efficient Works Fast Unit Test Continuous Integration Continuous Delivery Easy to Change Version Control Deployment Pipeline Dependency Injection Single Responsibility Object Oriented Functional Recursive Interface Reliability reusability Monitoring Branching Strategy Error Handling Cyclomatic Complexity Communication Protocol Package Management readability Distributed System flexibility Release Management Merging Conflict etc…

Objective for this talk:
Clean your code

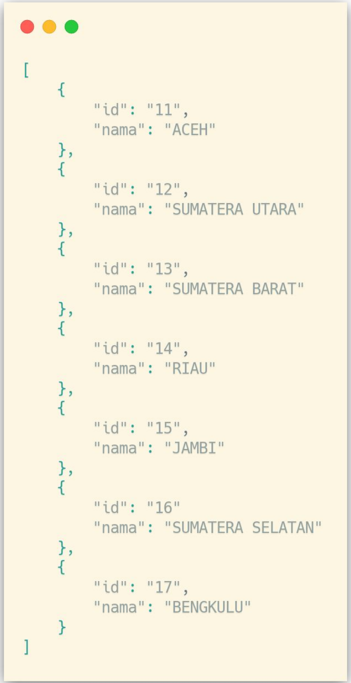
What is Clean Code?

Can you spot the bug?



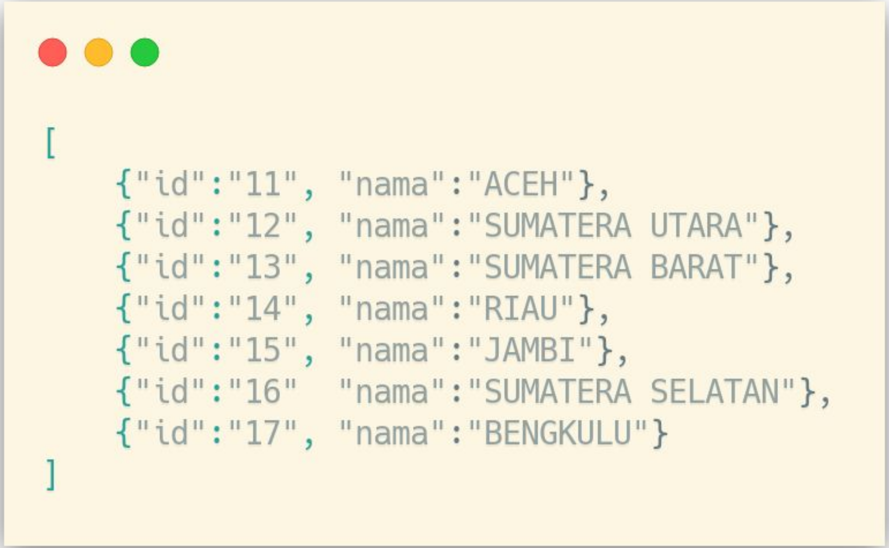
```
[{"id": "11", "nama": "ACEH"}, {"id": "12", "nama": "SUMATERA UTARA"},
{"id": "13", "nama":"SUMATERA BARAT"}, {"id": "14", "nama": "RIAU"}, {"id":
"15", "nama": "JAMBI"}, {"id": "16" "nama": "SUMATERA SELATAN"}, {"id":
"17", "nama": "BENGKULU"}]
```

Can you spot the bug now?

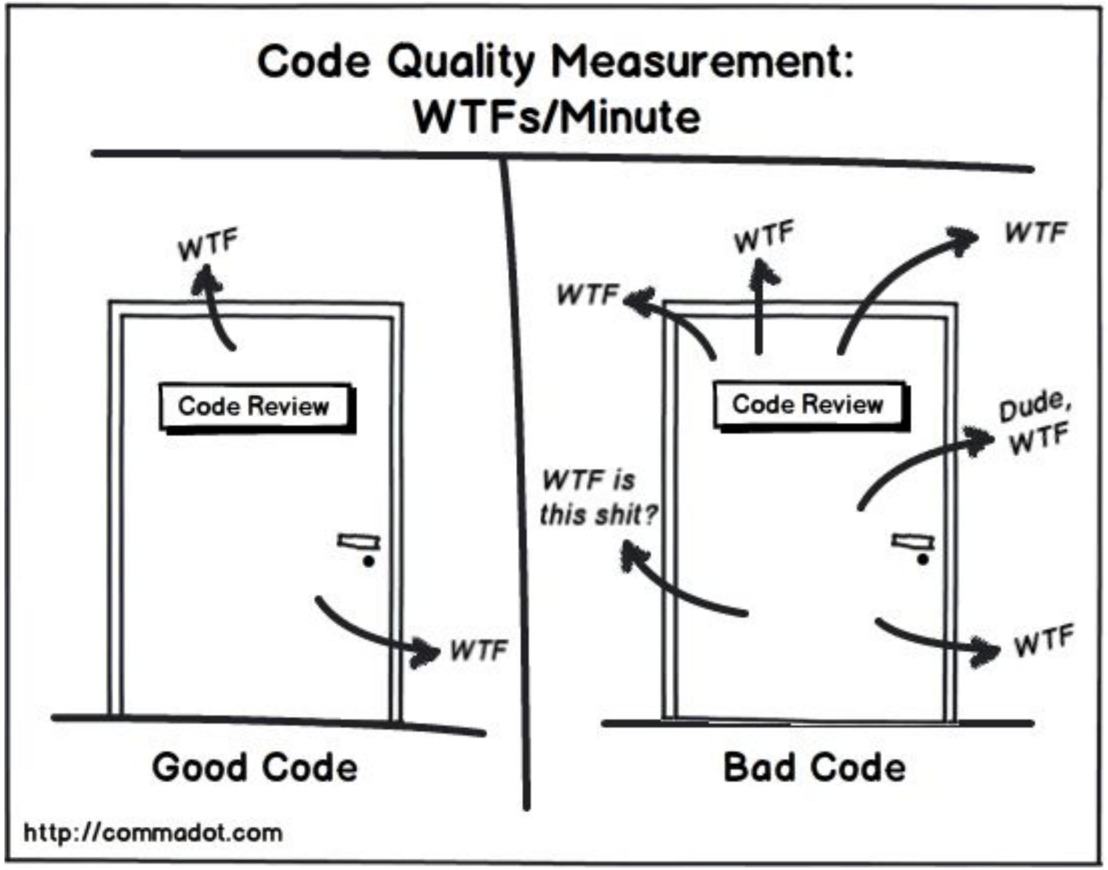


```
[
    {
        "id": "11",
        "nama": "ACEH"
    },
    {
        "id": "12",
        "nama": "SUMATERA UTARA"
    },
    {
        "id": "13",
        "nama": "SUMATERA BARAT"
    },
    {
        "id": "14",
        "nama": "RIAU"
    },
    {
        "id": "15",
        "nama": "JAMBI"
    },
    {
        "id": "16"
        "nama": "SUMATERA SELATAN"
    },
    {
        "id": "17",
        "nama": "BENGKULU"
    }
]
```

Can you spot the bug yet?



```
[
    {"id":"11", "nama":"ACEH"},
    {"id":"12", "nama":"SUMATERA UTARA"},
    {"id":"13", "nama":"SUMATERA BARAT"},
    {"id":"14", "nama":"RIAU"},
    {"id":"15", "nama":"JAMBI"},
    {"id":"16"  "nama":"SUMATERA SELATAN"},
    {"id":"17", "nama":"BENGKULU"}
]
```

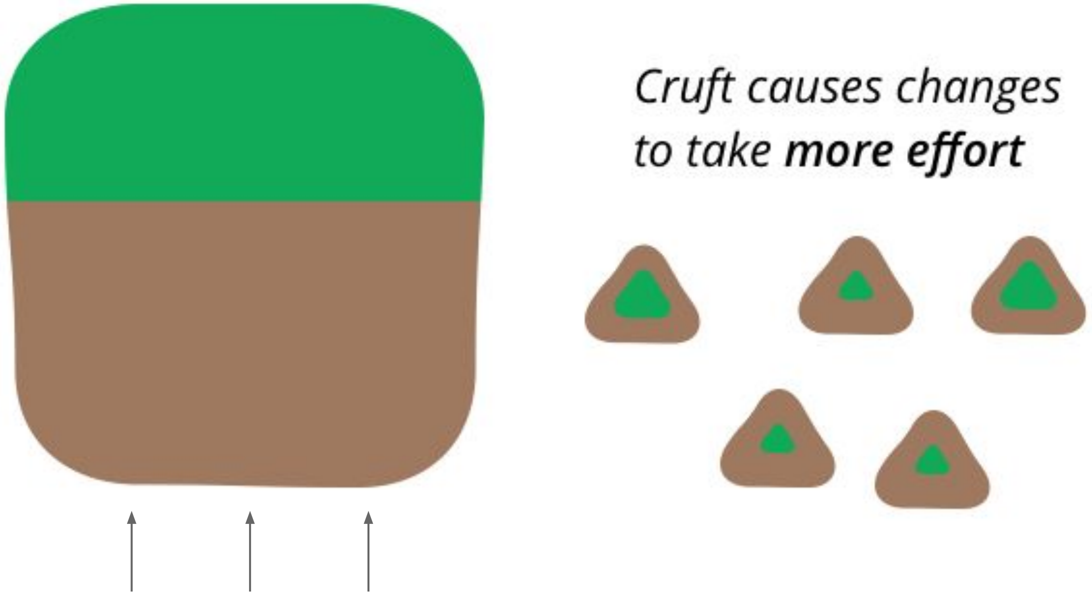

Code Quality Measurement:
WTFs/Minute
WTF
Code Review
WTF
Good Code
WTF
WTF
WTF
WTF
Code Review
Dude, WTF
WTF is this shit?
WTF
Bad Code
http://commadot.com

source: [1]

Why do we need Clean Code?

Any software system has a certain amount of ***essential*** *complexity required to do its job...*

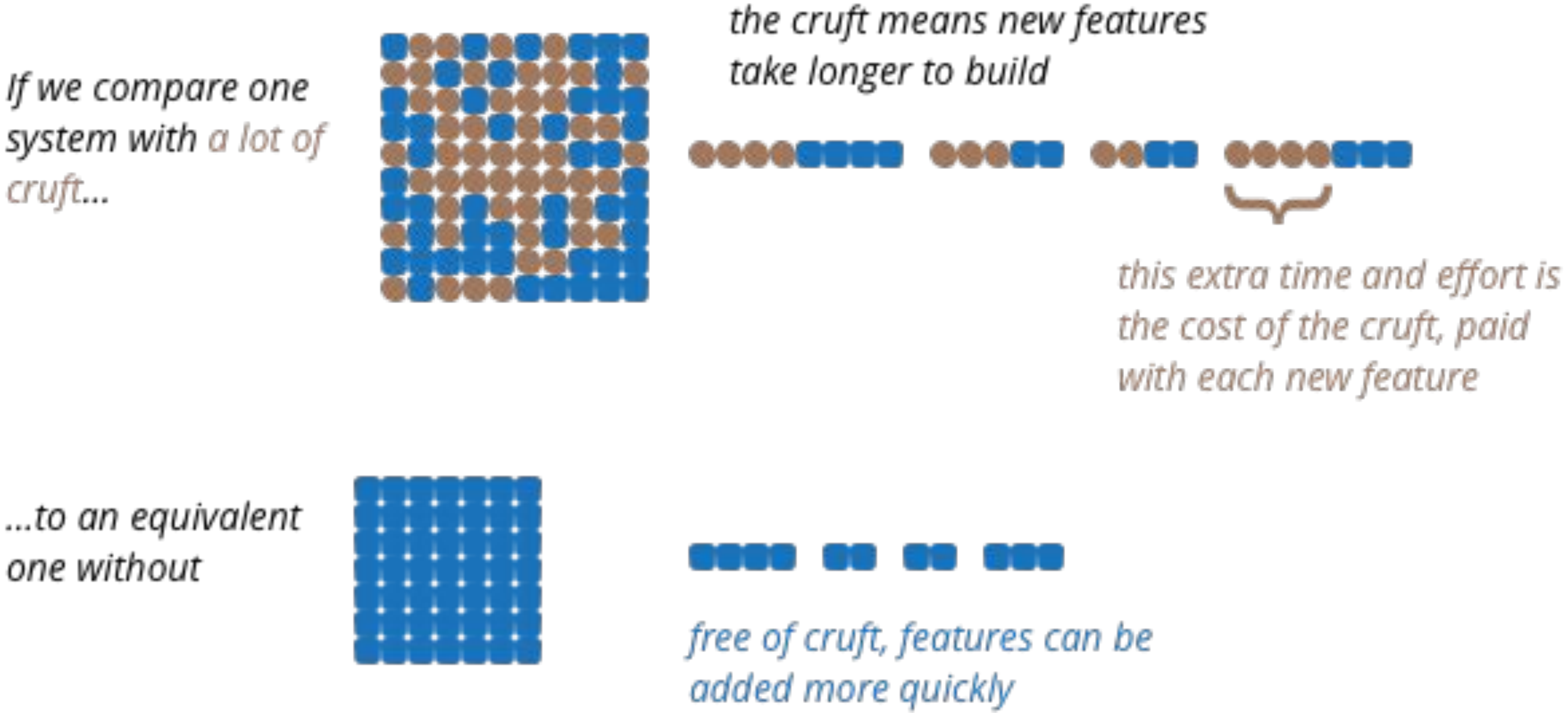
... but most systems contain ***cruft*** *that makes it harder to understand.*



Cruft causes changes to take ***more effort***

The technical debt metaphor treats the cruft as a debt, whose interest payments are the extra effort these changes require.

source: [2]


If we compare one system with a lot of cruft...

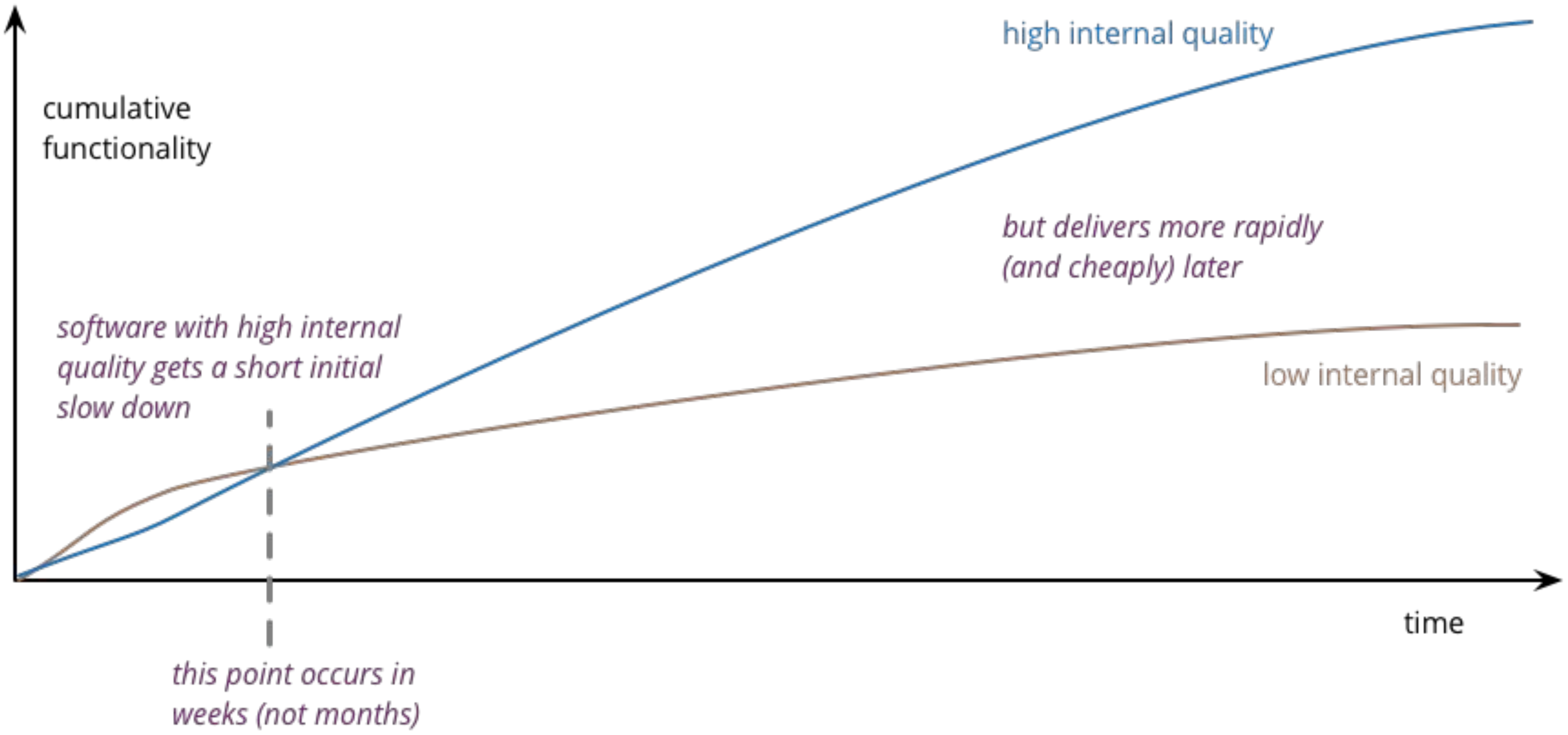
the cruft means new features take longer to build

this extra time and effort is the cost of the cruft, paid with each new feature

...to an equivalent one without

free of cruft, features can be added more quickly

source: [2]


high internal quality
cumulative functionality
but delivers more rapidly (and cheaply) later
software with high internal quality gets a short initial slow down
low internal quality
time
this point occurs in weeks (not months)

source: [2]

Code Smell

Bloaters

- Long method
- Large Class
- Primitive Obsession
- Long Parameter List
- Data Clumps

source [link]

source: [3][4]

Abusers

- Switch Statements
- Temporary Field
- Refused Bequest
- Alternative Classes with Different Interfaces

source [link]

source: [3][4]

Change Preventers

- Divergent Change
- Shotgun Surgery
- Parallel Inheritance Hierarchies

source [link]

source: [3][4]

Dispensables

- Duplicate Code
- Lazy Class
- Data Class
- Speculative Generality

source [link]

source: [3][4]

Couplers

- Feature Envy
- Inappropriate Intimacy
- Message Chains
- Middle Man

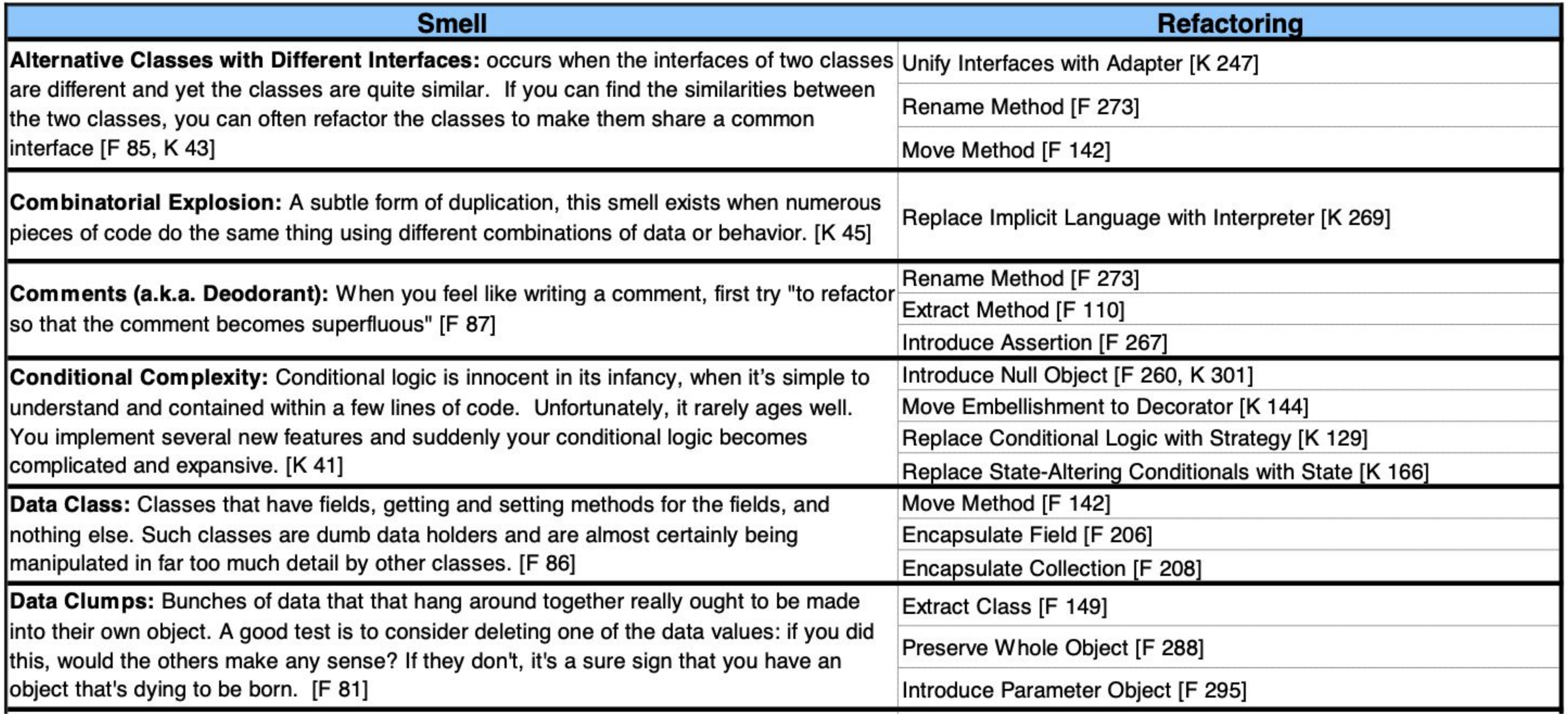
source [link]

source: [3][4]

Static Code Analyzer!

- https://github.com/golangci/awesome-go-linters
- https://github.com/prontolabs/pronto
- Google it yourself “<your language of choice i.e. Golang> code smell linters”

Refactoring pattern for each Code Smell



Smell	Refactoring
Alternative Classes with Different Interfaces: occurs when the interfaces of two classes are different and yet the classes are quite similar. If you can find the similarities between the two classes, you can often refactor the classes to make them share a common interface [F 85, K 43]	Unify Interfaces with Adapter [K 247]
	Rename Method [F 273]
	Move Method [F 142]
Combinatorial Explosion: A subtle form of duplication, this smell exists when numerous pieces of code do the same thing using different combinations of data or behavior. [K 45]	Replace Implicit Language with Interpreter [K 269]
Comments (a.k.a. Deodorant): When you feel like writing a comment, first try "to refactor so that the comment becomes superfluous" [F 87]	Rename Method [F 273]
	Extract Method [F 110]
	Introduce Assertion [F 267]
Conditional Complexity: Conditional logic is innocent in its infancy, when it's simple to understand and contained within a few lines of code. Unfortunately, it rarely ages well. You implement several new features and suddenly your conditional logic becomes complicated and expansive. [K 41]	Introduce Null Object [F 260, K 301]
	Move Embellishment to Decorator [K 144]
	Replace Conditional Logic with Strategy [K 129]
	Replace State-Altering Conditionals with State [K 166]
Data Class: Classes that have fields, getting and setting methods for the fields, and nothing else. Such classes are dumb data holders and are almost certainly being manipulated in far too much detail by other classes. [F 86]	Move Method [F 142]
	Encapsulate Field [F 206]
	Encapsulate Collection [F 208]
Data Clumps: Bunches of data that that hang around together really ought to be made into their own object. A good test is to consider deleting one of the data values: if you did this, would the others make any sense? If they don't, it's a sure sign that you have an object that's dying to be born. [F 81]	Extract Class [F 149]
	Preserve Whole Object [F 288]
	Introduce Parameter Object [F 295]

more on: [3]

Credits

- [1] Commadot.com for WTFs/minutes
- [2] https://martinfowler.com/articles/is-quality-worth-cost.html
- [3] https://www.industriallogic.com/img/blog/2005/09/smellstorefactorings.pdf
- [4] https://sourcemaking.com
- Fowler, Martin. Refactoring: Improving the Design of Existing Code
- Kerievsky, Joshua. Refactoring to Patterns

QnA

Do you have any question?

We’re hiring

mekari.com/careers

Follow Mekari on Instagram @lifeatmekari and LinkedIn “Mekari” for info on our next events and updated job vacancy (monthly)