Refactoring and code smells

Martin Osovsky
Y Soft

The outline

* What is good code?
* What is refactoring?
* The importance of testing
* When, why a where to refactor?
* Examples

Important people

* @martinfowler
* @unclebobmartin
* @Bertrand_Meyer
* @ploeh
* @KevlinHenney
* @tastapod

Important books

* Robert C. Martin : Clean Code
* Robert C. Martin : Agile Principles, Patterns, and Practices in C#
* Martin Fowler : Refactoring: Improving the Design of Existing Code
* Joshua Kerievsky : Refactoring to patterns
* Michael Feathers : Working effectively with Legacy Code
* Garry M. Hall : Adaptive Code via C#
* Gerard Mezsaros : xUnit Test Patterns, Refactoring Test Code

We have already covered…

* 4 Rules of Simple Design
* Unit Testing

SOLID Principles

* Single Responsibility
* Open / Closed
* Liskov Substitution Principle
* Interface Segregation
* Dependency Inversion

Code Smells

1. different abstraction levels (not top down - mixed, skipping levels, mixing levels in one method)

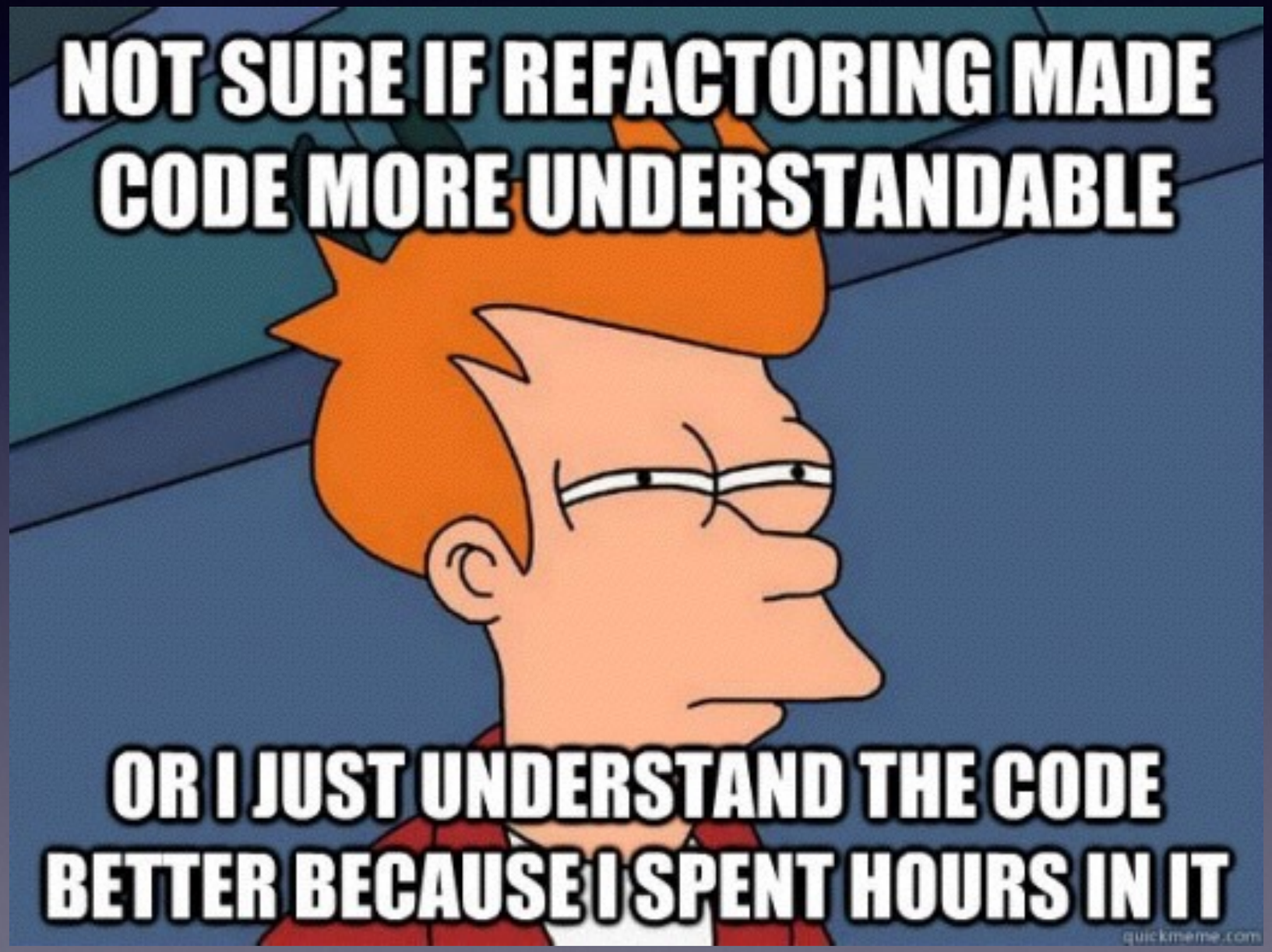
2. circular dependencies (between classes - mother of all tight couplings)

3. low cohesion (god classes, script/program wrapped as a class)

4. bad naming (incosistent, non-clear terminology, non-standard terminology meaningless names, abbreviations, hungarian notation, pleonasms, FactoryClass, IDisposable, …Exception)

5. Pokemon smell - catch them all, exception abuse

REFACTORING



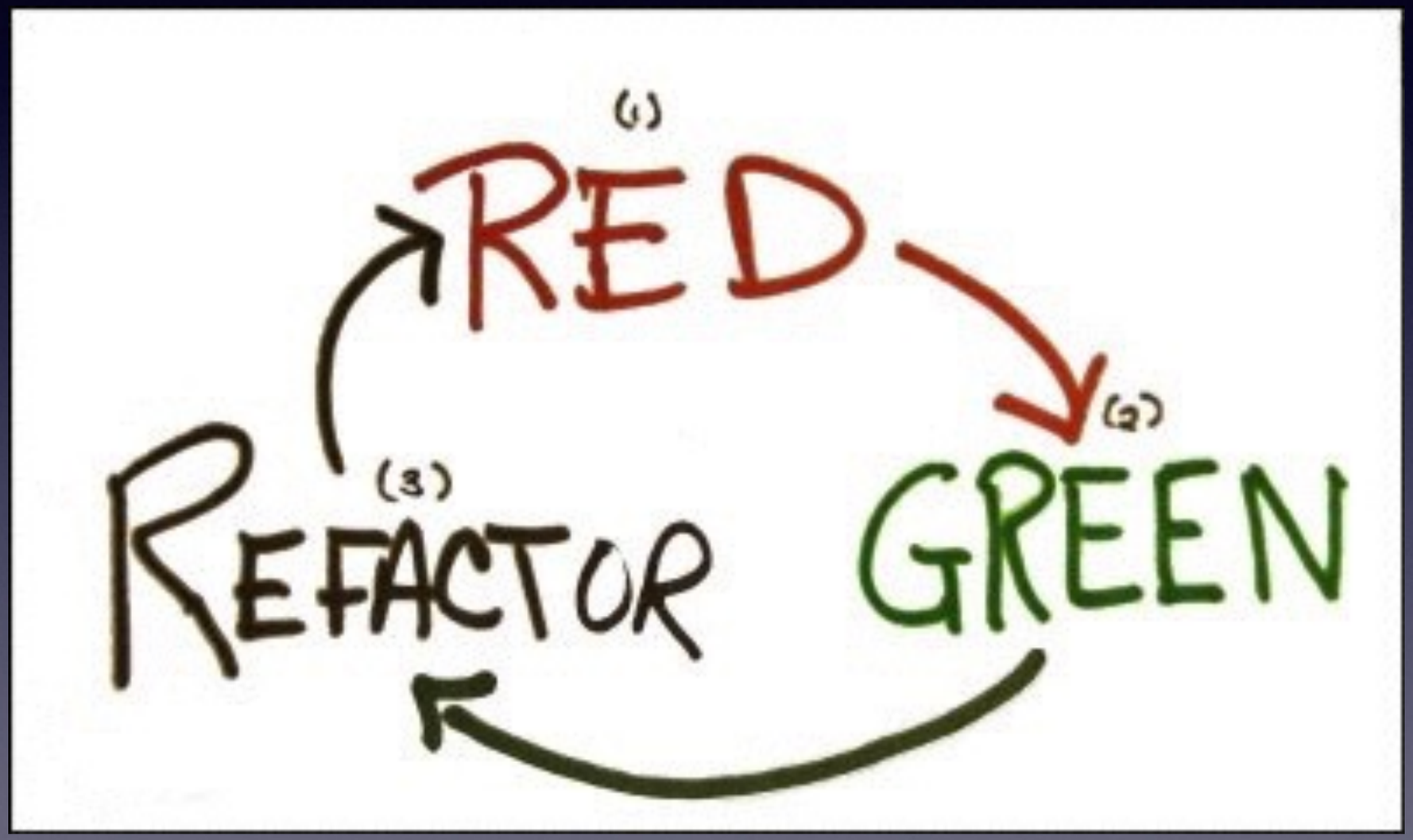
What is refactoring?

* Controlled change in code that doesn't change its external (published) behaviour but improves internal structure

* Refactoring vs redesign

* Refactoring in the strict sense

When to refactor?



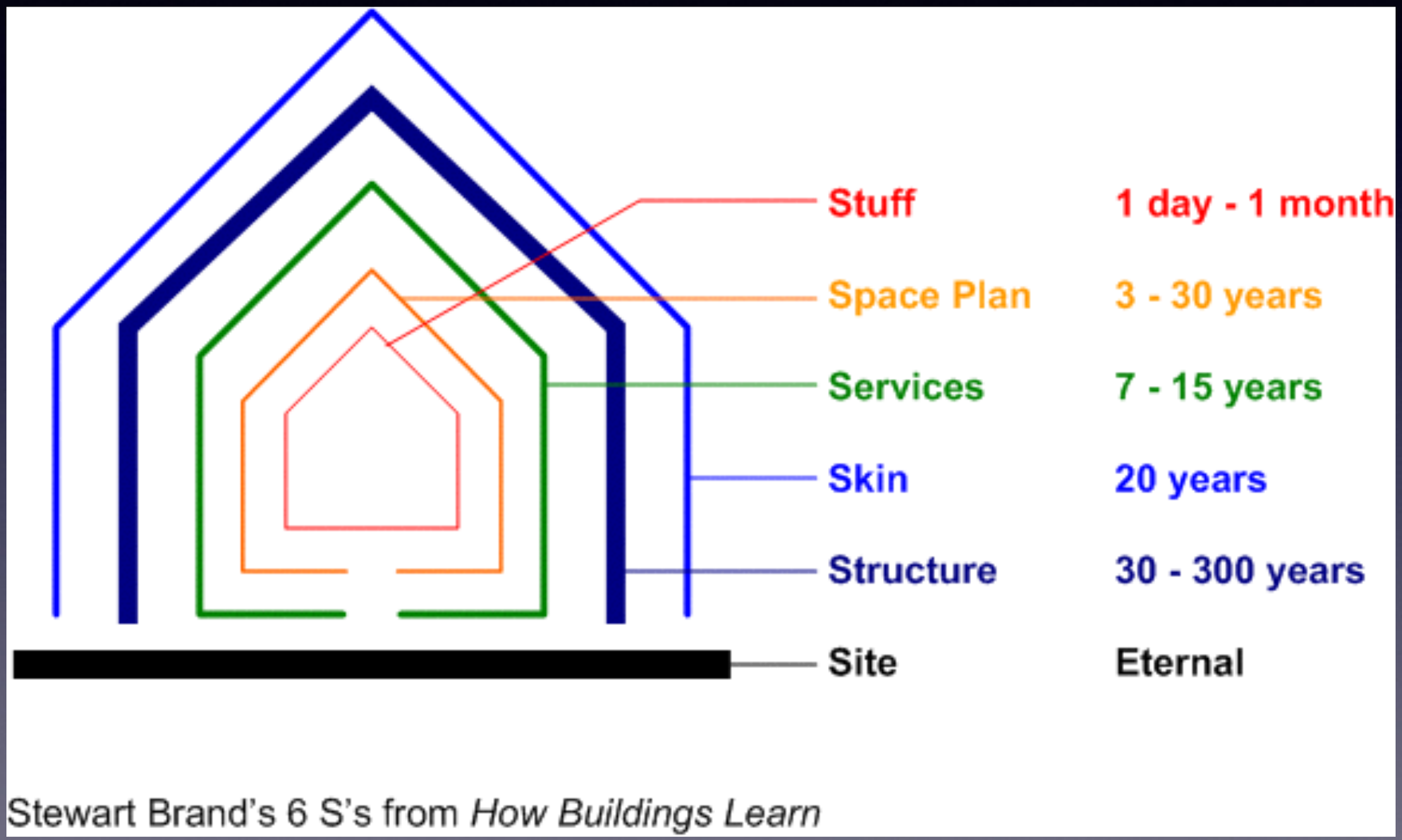
When to refactor?

* When you have the refactoring hat on your head!
* As part of the routine (e.g. TDD)
* After you find weak code (boy scout rule)
* Before you need to introduce code of a new feature (or a new technology like IoC container)
* Long term planned refactoring
    * you need to have a plan and know what is the final state

When to refactor?

Only when it leads to faster delivery and better maintenance. Clean code is means to this end.

Where to refactor?



Stewart Brand's 6 S's from *How Buildings Learn*

Where to refactor

* The application same as a building has layers that have different
    * cost of change (outer walls = new building)
    * rate of change
* Architecture = the most slow and costly parts of the system (outer walls, foundations)

How?

* Use IDE all the time (even when renaming!)
* Run tests before and after
* Boundary tests (testing published interfaces) should stay green
* know most common refactorings (extract …, rename, move, introduce) - learn to use them as part of your IDE mastery

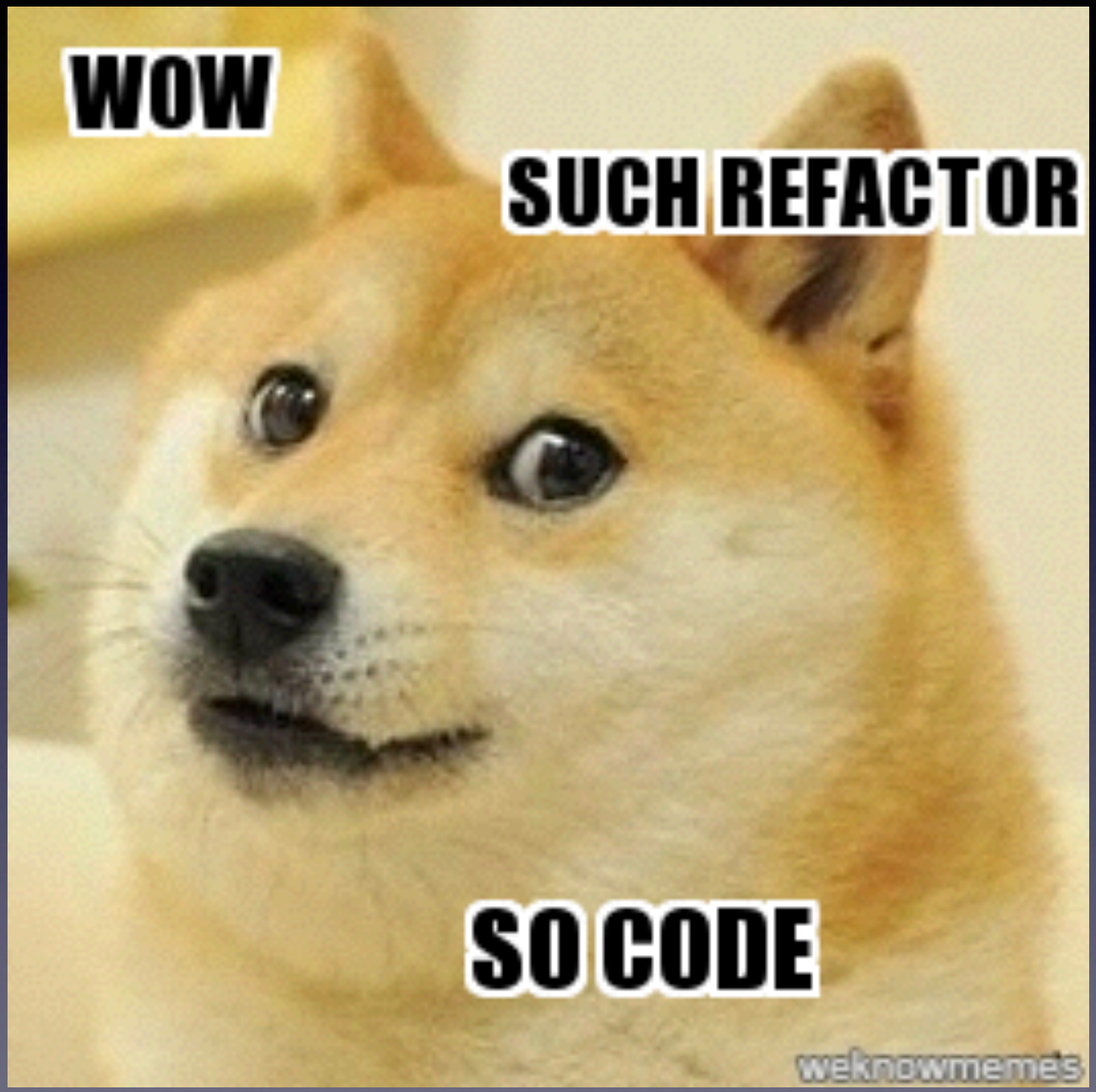
And most importantly

Code and discuss your code with others and learn from the best (github is full of great code)

Follow the guys form the second slide

Even more importantly

Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.


WOW
SUCH REFACTOR
SO CODE
weknowmemes