Refactoring 102

or: How I learned to stop worrying and started to remove code

Alex Poovathingal

Gozefo.com

January 11, 2019

Introduction

Disclaimer - The author shall not be held liable for any issuesof whatever nature which may arise as a result of following these advices.

What is Refactoring

What is Refactoring

- Restructure for the better without changing behaviour

What is Refactoring

- Restructure for the better without changing behaviour
- Remove code smells

What is Refactoring

- Restructure for the better without changing behaviour
- Remove code smells
- Reduce technical debt

What is Refactoring

- Restructure for the better without changing behaviour
- Remove code smells
- Reduce technical debt
- Without modifying testcases

What is Refactoring

- Restructure for the better without changing behaviour
- Remove code smells
- Reduce technical debt
- Without modifying testcases
- "Leave this world a little better than you found it." - Boy Scout Rule

When to Refactor and things to keep in mind

- “Don’t stop coding when it starts to work”

When to Refactor and things to keep in mind

- "Don't stop coding when it starts to work"
- Yucky code

When to Refactor and things to keep in mind

- “Don’t stop coding when it starts to work”
- Yucky code
- comprehending difficult to understand code should never be the last step

When to Refactor and things to keep in mind

- “Don’t stop coding when it starts to work”
- Yucky code
- comprehending difficult to understand code should never be the last step
- ”Does this code spark joy” - Marie Kondo

When to Refactor and things to keep in mind

- “Don’t stop coding when it starts to work”
- Yucky code
- comprehending difficult to understand code should never be the last step
- ”Does this code spark joy” - Marie Kondo
- Why?? Refactoring is good economics - Is refactoring wasteful rework?

When to Refactor and things to keep in mind

- “Don’t stop coding when it starts to work”
- Yucky code
- comprehending difficult to understand code should never be the last step
- ”Does this code spark joy” - Marie Kondo
- Why?? Refactoring is good economics - Is refactoring wasteful rework?
- Refactor v/s features - do any one at a time - not both

When to Refactor and things to keep in mind

- “Don’t stop coding when it starts to work”
- Yucky code
- comprehending difficult to understand code should never be the last step
- ”Does this code spark joy” - Marie Kondo
- Why?? Refactoring is good economics - Is refactoring wasteful rework?
- Refactor v/s features - do any one at a time - not both
- Refactor - be ready to throw away code. ”Don’t be emotionally attached to your code”

When to Refactor and things to keep in mind

- “Don’t stop coding when it starts to work”
- Yucky code
- comprehending difficult to understand code should never be the last step
- ”Does this code spark joy” - Marie Kondo
- Why?? Refactoring is good economics - Is refactoring wasteful rework?
- Refactor v/s features - do any one at a time - not both
- Refactor - be ready to throw away code. ”Don’t be emotionally attached to your code”
- Workflow - small improvements (always in same direction)

When to Refactor and things to keep in mind

- “Don’t stop coding when it starts to work”
- Yucky code
- comprehending difficult to understand code should never be the last step
- ”Does this code spark joy” - Marie Kondo
- Why?? Refactoring is good economics - Is refactoring wasteful rework?
- Refactor v/s features - do any one at a time - not both
- Refactor - be ready to throw away code. ”Don’t be emotionally attached to your code”
- Workflow - small improvements (always in same direction)
- File renames or package changes always in different PR and when no one is working on those files

Plurals

```
Integer productsInCart = 2;
```

Plurals

```
Integer productsInCart = 2;

List<CartProduct> productsInCart = [product1, product2];
```

Plurals

```
Integer productsInCart = 2;

List<CartProduct> productsInCart = [product1, product2];

Better - cartProductCount, cartProductList,
cartProducts (?)
```

Plurals

```
Integer productsInCart = 2;

List<CartProduct> productsInCart = [product1, product2];

Better - cartProductCount, cartProductList,
cartProducts (?)

products v/s productList
productNames, productsNames, productsName, productNameList
```

Proper Names

```
ProductParts
ProductParts.id and ProductParts.productPartId
```

Proper Names

```
ProductParts
ProductParts.id and ProductParts.productPartId

String productPartId = productPart.getId();
String productPartId = productPart.getProductPartId();
```

Mysql

- Plural table Name (Controversial)
- Singular entity
- updated_on vs updatedOn
- Worst case - learn language

Removing Code

- Less code, less bugs. No code, ...

Removing Code

- Less code, less bugs. No code, ...
- Unused files, unused functions

Removing Code

- Less code, less bugs. No code, ...
- Unused files, unused functions
- Unused APIs

Visual

- Code consistency throughout the project

Visual

- Code consistency throughout the project
- Arguments on each line

Visual

- Code consistency throughout the project
- Arguments on each line
- Easier if condition

Visual

- Code consistency throughout the project
- Arguments on each line
- Easier if condition
- Readability always

Visual

- Code consistency throughout the project
- Arguments on each line
- Easier if condition
- Readability always
- Intellij warnings

Visual

- Code consistency throughout the project
- Arguments on each line
- Easier if condition
- Readability always
- Intellij warnings
- Line length - 150, function length, class length

Pull Request Etiquettes

- git diff

Pull Request Etiquettes

- git diff
- Make life easier for reviewer.

Pull Request Etiquettes

- git diff
- Make life easier for reviewer.
- Lesser lines - what do each line do

Pull Request Etiquettes

- git diff
- Make life easier for reviewer.
- Lesser lines - what do each line do
- Blank lines, format issues, unrelated changes

Bug fixing Workflow

- Write failing test. Make sure it fails
- Make your code change.
- Test should pass now

Bug fixing Workflow

- Write failing test. Make sure it fails
- Make your code change.
- Test should pass now

Tests should be

- Concise
- Readable
- Isolated

Unsolicited Advice (Polished Rants)

- Upfront Design v/s evolutionary design

Unsolicited Advice (Polished Rants)

- Upfront Design v/s evolutionary design
- If your code is not unit testable, it is bad code

Unsolicited Advice (Polished Rants)

- Upfront Design v/s evolutionary design
- If your code is not unit testable, it is bad code
- Small changes every day take you to big refactoring

Unsolicited Advice (Polished Rants)

- Upfront Design v/s evolutionary design
- If your code is not unit testable, it is bad code
- Small changes every day take you to big refactoring
- Compilation errors are good

Unsolicited Advice (Polished Rants)

- Upfront Design v/s evolutionary design
- If your code is not unit testable, it is bad code
- Small changes every day take you to big refactoring
- Compilation errors are good
- Easy to read, understand, navigate >>>performance, design pattern, world peace, or any other thing

Unsolicited Advice (Polished Rants)

- Upfront Design v/s evolutionary design
- If your code is not unit testable, it is bad code
- Small changes every day take you to big refactoring
- Compilation errors are good
- Easy to read, understand, navigate >>>performance, design pattern, world peace, or any other thing
- Not everything needs to scale - MOST things doesn't need to scale

Unsolicited Advice (Polished Rants)

- Upfront Design v/s evolutionary design
- If your code is not unit testable, it is bad code
- Small changes every day take you to big refactoring
- Compilation errors are good
- Easy to read, understand, navigate >>>performance, design pattern, world peace, or any other thing
- Not everything needs to scale - MOST things doesn't need to scale
- Code is better than infrastructure

Any more points?