Emily Bache

TECH COACH EB CONSULTANT

Reading Legacy Code Effectively

October 2024

Talk Description - 25 minutes

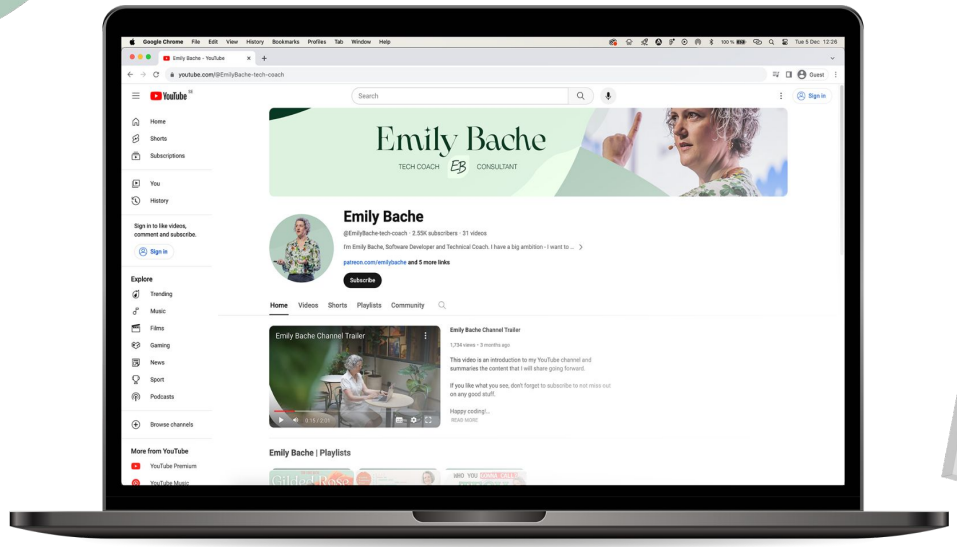
Reading code is sometimes so difficult that developers will simply give up and complain loudly that they need to rewrite the code from scratch because it is so incomprehensible. (Preferably the rewrite will be in a fancy new language or framework they have just heard about at a conference). I can totally understand the sentiment but it's almost always a mistake to rewrite valuable working code from scratch. On the other hand it's true that legacy code can be very challenging to read, with long sections of complex logic and obtuse names. Unfortunately despite it being a crucial skill - we spend more time reading code than writing it - most people are never taught code reading strategies.

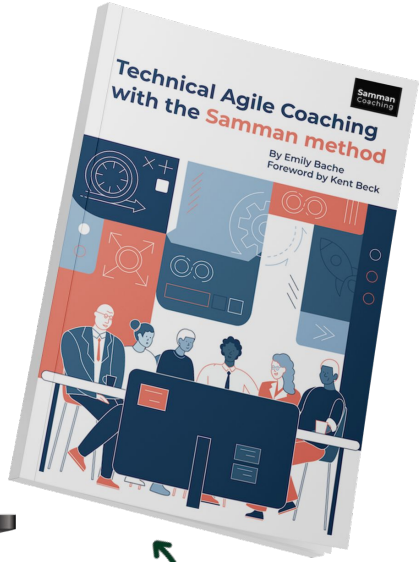
In this presentation I will go through some of the techniques I use that are effective even when you have legacy code. My goal is always to quickly get to the point where I understand enough context to find the part I need to change, and to do that safely. I always want to leave the code in a more readable state than I found it, to help the next person. I'll discuss "Scanning", effective use of your code editor tooling, and "Naming as a Process". We'll go through a code reading training exercise together and I'll show lots of examples. Next time someone claims some code is incomprehensible you might be able to help them avoid that expensive and risky rewrite.

Emily Bache

TECH COACH EB CONSULTANT

YouTube Channel





Book

@emilybache

emily@bacheconsulting.com

Situation

Reading Code is Hard

Sensemaking and Orientation

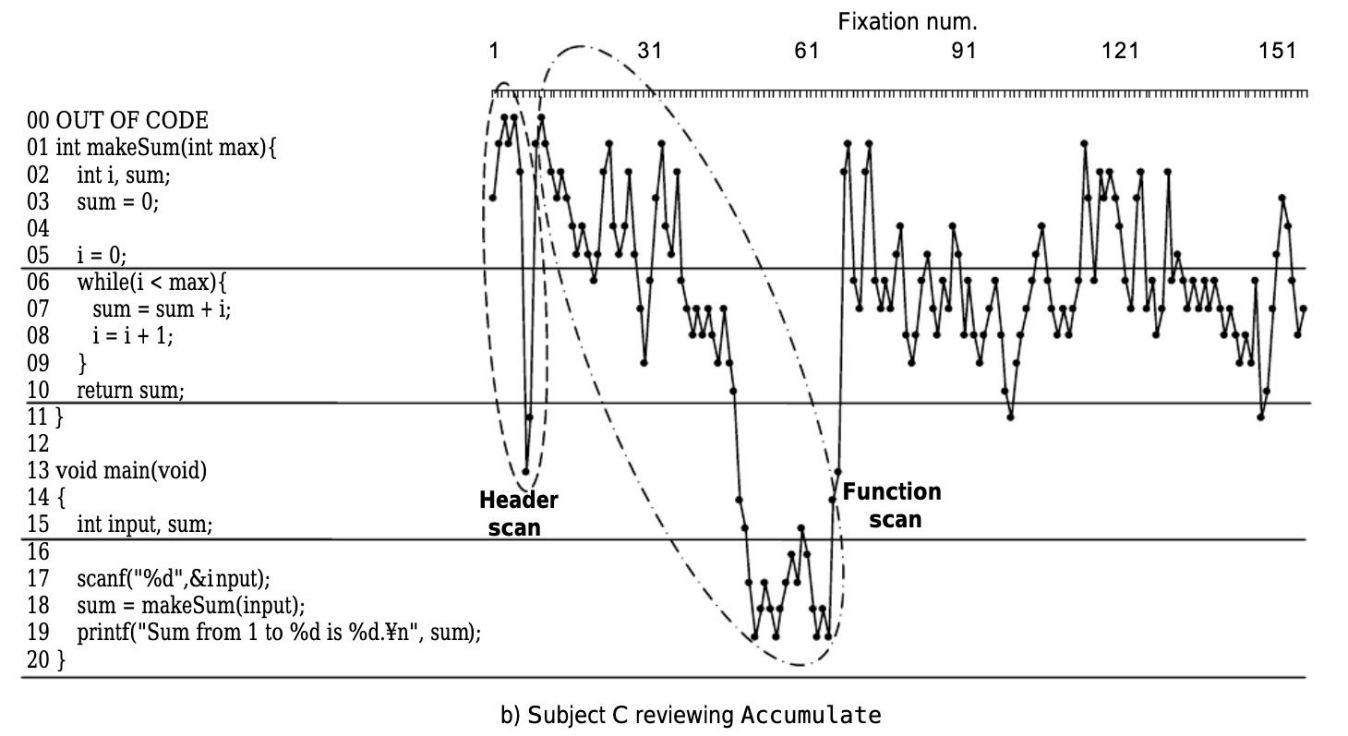
Not like reading a novel

Goals

- Quickly understand enough context
- Find a good place to start making changes

```
00 OUT OF CODE
01 int makeSum(int max){
02    int i, sum;
03    sum = 0;
04
05    i = 0;
06    while(i < max){
07      sum = sum + i;
08      i = i + 1;
09    }
10    return sum;
11 }
12
13 void main(void)
14 {
15    int input, sum;
16
17    scanf("%d",&input);
18    sum = makeSum(input);
19    printf("Sum from 1 to %d is %d.¥n", sum);
20 }
```

Scanning



b) Subject C reviewing `Accumulate`

"Analyzing individual performance of source code review using reviewers' eye movement" by Uwano et al, 2006

Experts and Novices

“Experts read code less linearly than novices.”

"Experts are better able to focus on the relevant source code elements than novices"

"one of the effective ways to improve skill acquisition is to cue visual attention of novices to the locations that experts attend [to]"

"Eye movements in code reading" by Busjahn, Teresa et al, 2015

Heuristics for Reading Code

- Scanning
- Quick Wins
- Scratch Refactoring
- Communicate what you learn

Scanning Practice

- I will slowly scroll through the code
- What do you see?

https://github.com/emilybache/Tennis-Refactoring-Kata

Scanning : Problems Detected

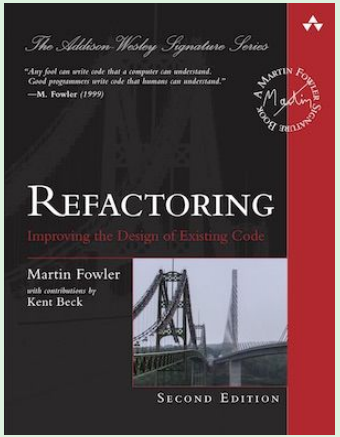
In the Tennis6 class:

- Long Method
- Code Paragraphs

Code Smells

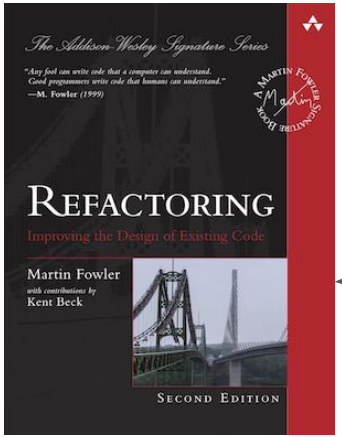
> “A code smell is a surface indication that usually corresponds to a deeper problem in the system”
>
> – Martin Fowler

- Quick to spot
- Doesn’t *always* indicate a problem
- Smells have names
- Smells suggest suitable refactorings



https://martinfowler.com/bliki/CodeSmell.html

Code Smells Have Names



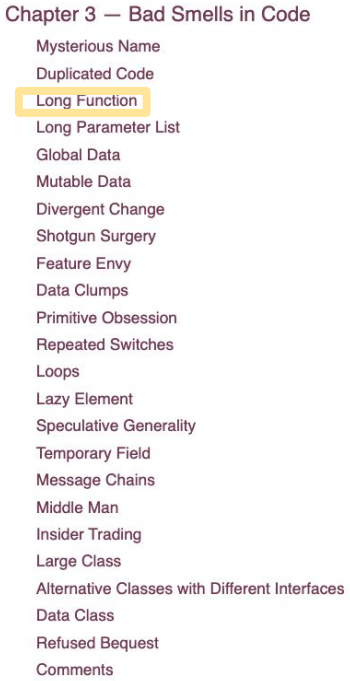
24 Named Code Smells

Chapter 3 — Bad Smells in Code

- Mysterious Name
- Duplicated Code
- Long Function
- Long Parameter List
- Global Data
- Mutable Data
- Divergent Change
- Shotgun Surgery
- Feature Envy
- Data Clumps
- Primitive Obsession
- Repeated Switches
- Loops
- Lazy Element
- Speculative Generality
- Temporary Field
- Message Chains
- Middle Man
- Insider Trading
- Large Class
- Alternative Classes with Different Interfaces
- Data Class
- Refused Bequest
- Comments

Long Function & Long Method

- Easy to spot via ‘scanning’
- Doesn’t *always* indicate a problem
- Several smaller methods would *often* be better



Code Paragraph

- Easy to spot via ‘scanning’
- Doesn’t *always* indicate a problem
- Extract and name as a method

Paragraph of Code

A paragraph is a section of code within a longer function or method that belongs together, does something specific, and might make sense to extract and name as a method. Giving the section of code a name and explicitly stating the arguments, dependencies and return value would almost certainly be easier to read and reason about than an unnamed section of code.

You can spot code paragraphs without reading the code in any detail, they have one or more of these visual characteristics:

- begin with a short comment explaining what the next lines of code do
- are in a code block (ie they are between a pair of curly braces or at the same indentation level)
- begin with a for, if, try, switch or while statement
- have a line of whitespace before and after
- a cluster of statements using the same variable name or recurring word

Sources

- This smell was contributed to this site by Emily Bache

Quick Wins

In the Tennis6 class:

- Long Method
- Code Paragraphs

Quick Wins:

- Extract Method

Scanning Practice

- I will slowly scroll through the code
- What do you see?

https://github.com/emilybache/GildedRose-Refactoring-Kata

Scanning : Problems Detected

In the GildedRose class:

- Long Method
- Deep Nesting
- IDE highlights

Quick Wins

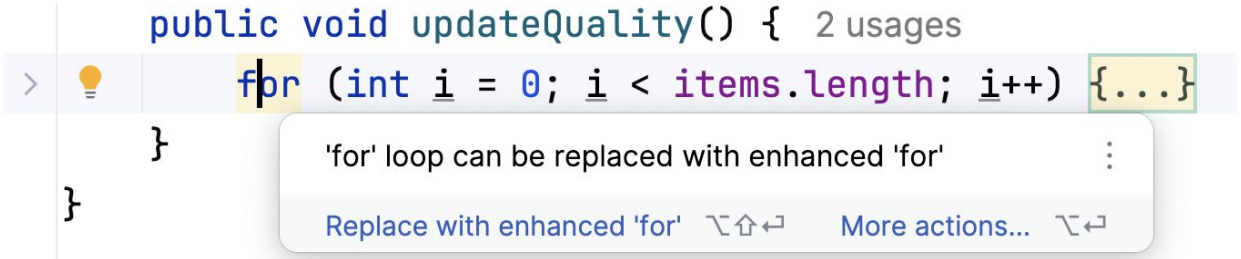
In the GildedRose class:

- Long Method
- Deep Nesting
- IDE highlights

Quick Wins:

- Fix IDE highlights
- (Maybe) Extract Method

Quick Wins



Quick Wins:

- Fix IDE highlights

Scanning Practice

- I will slowly scroll through the code
- What do you see?

https://github.com/emilybache/Tennis-Refactoring-Kata

Scanning : Problems Detected

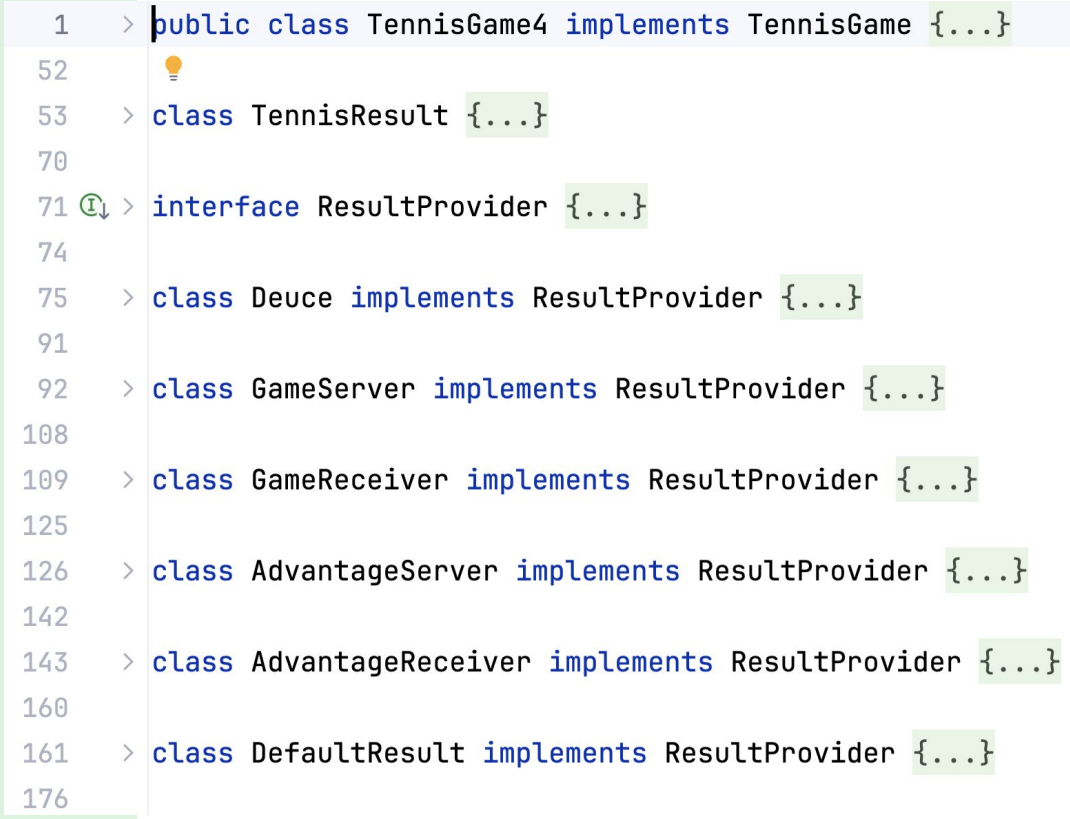
In the Tennis4 class:

- Long line of code
- Lots of small things

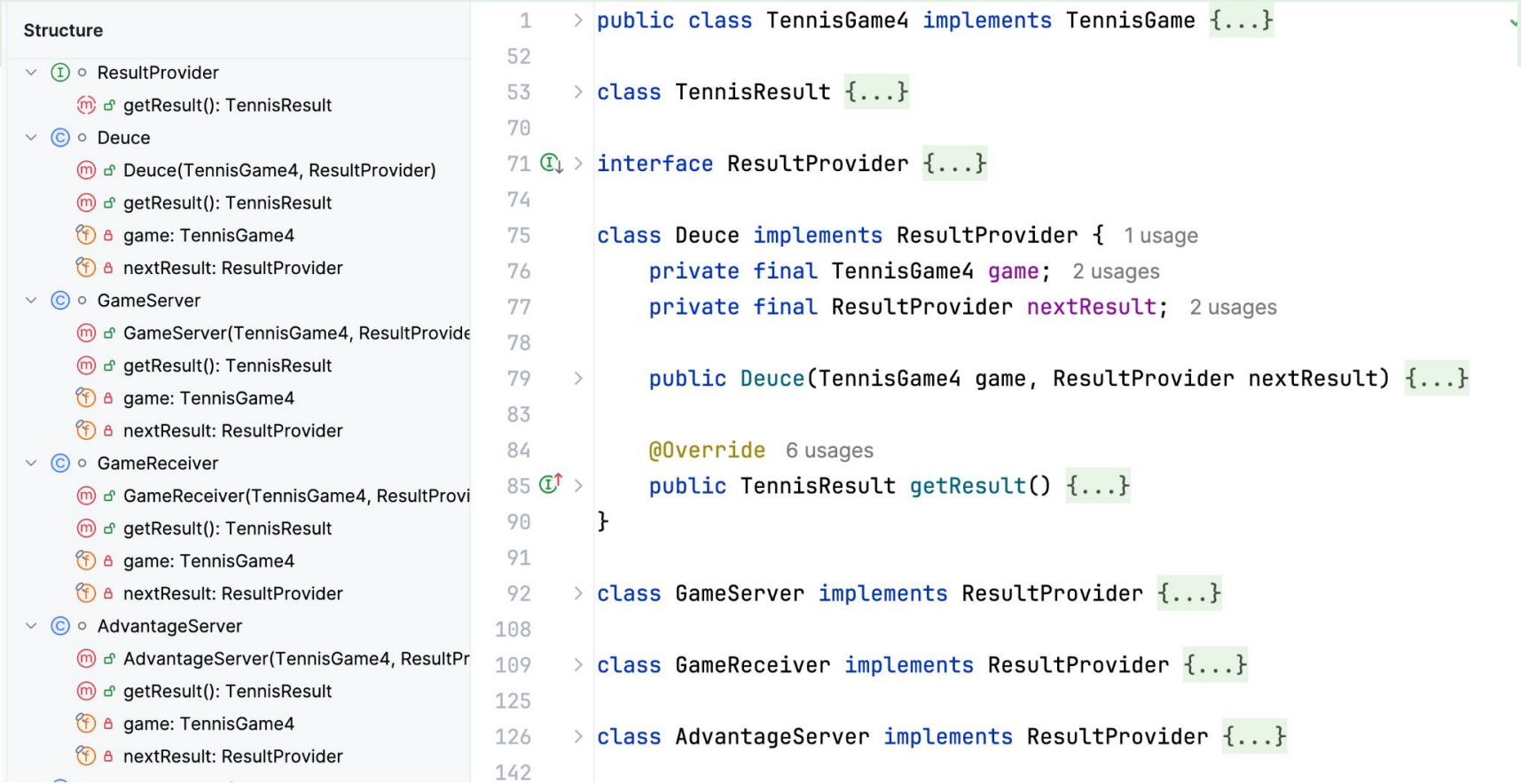
```
21          @java.lang.Override
22          public String getScore() {
23              TennisResult result = new Deuce(
24                      this, new GameServer(
25                              this, new GameReceiver(
26                                      this, new AdvantageServer(
27                                              this, new AdvantageReceiver(
28                                                      this, new DefaultResult(this)))))).getResult();
29              return result.format();
30          }
```

Code Folding

- Your IDE can ‘fold’ the code
- Quickly understand the code’s structure
- Look for patterns



Structure View



Code Folding a long method

```
3   class GildedRose {  4 usages
4       Item[] items;  36 usages
5
6       public GildedRose(Item[] items) { this.items = items; }
9
10      public void updateQuality() {...}
62  }
```

- Top level structure of Gilded Rose is not very interesting

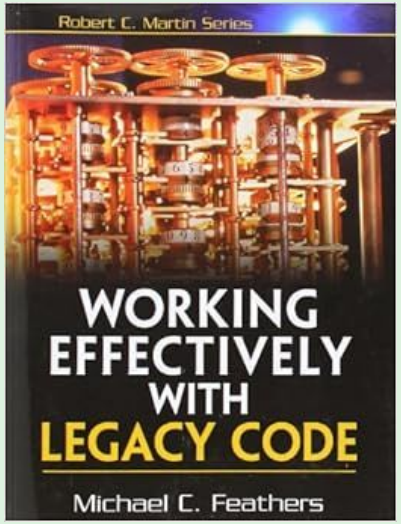
Code Folding : structure of a long method

```
3   class GildedRose {  4 usages
4       Item[] items;  36 usages
5
6       public GildedRose(Item[] items) { this.items = items; }
9
10      public void updateQuality() {  2 usages
11          for (int i = 0; i < items.length; i++) {
12              if (!items[i].name.equals("Aged Brie")
13                      && !items[i].name.equals("Backstage passes to a TAFKAL80ETC concert")) {...} else {...}
38
39              if (!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {...}
42
43              if (items[i].sellIn < 0) {...}
60          }
61      }
62  }
```

Scratch Refactoring

“One of the best techniques for learning about code is refactoring”

– Michael Feathers



You understand code better if you (try to) change it

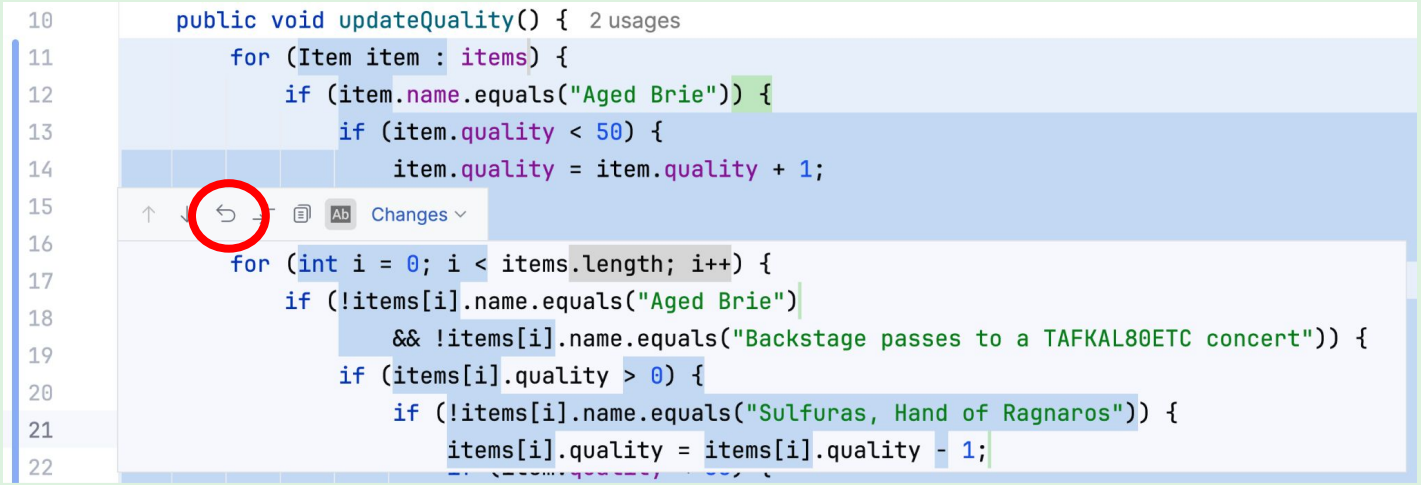
Scratch Refactoring

- Refactor, restructure, move things around…
- Learn how the code is put together
- Feedback from compiler & tests

```
10      public void updateQuality() {  2 usages
11          for (Item item : items) {
12              if (item.name.equals("Aged Brie")) {
13                  if (item.quality < 50) {
14                      item.quality = item.quality + 1;
15
16                  }
17              } else if (item.name.equals("Backstage passes to a TAFKAL80ETC concert")) {
18                  if (item.quality < 50) {
19                      item.quality = item.quality + 1;
20
21                      if (item.sellIn < 11) {
22                          if (item.quality < 50) {
```

Scratch Refactoring

- Refactor, restructure, move things around…
- Learn how the code is put together
- Feedback from compiler & tests



Throw away your changes

Communicate

Put what you learnt into the code & make it more readable for the next person

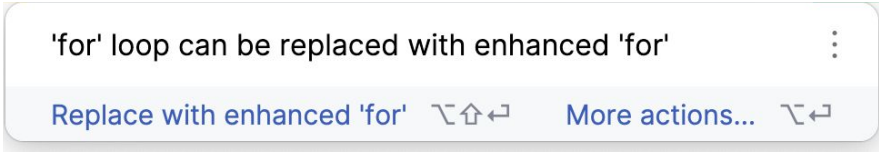
- Rename elements
- Extract methods & name chunks of code

Naming as a Process

https://www.digdeeproots.com/articles/on/naming-process/

Emily Bache

TECH COACH EB CONSULTANT

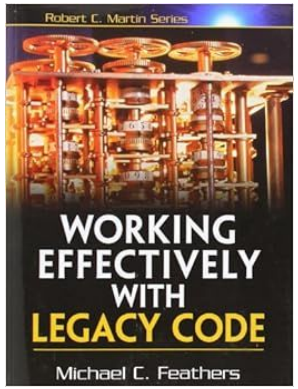


Quick Wins

```
public void updateQuality() {
    for (int i = 0; i < items.length; i++) {
        if (!items[i].name.equals("Aged Brie")
                && !items[i].name.equals("Backstage passes to a TAFKAL80ETC concert")) {
            if (items[i].quality > 0) {
                if (!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {
                    items[i].quality = items[i].quality - 1;
                }
            }
        } else {
            if (items[i].quality < 50) {
                items[i].quality = items[i].quality + 1;

                if (items[i].name.equals("Backstage passes to a TAFKAL80ETC concert")) {
                    if (items[i].sellIn < 11) {
                        if (items[i].quality < 50) {
                            items[i].quality = items[i].quality + 1;
                        }
                    }

                    if (items[i].sellIn < 6) {
                        if (items[i].quality < 50) {
                            items[i].quality = items[i].quality + 1;
                        }
                    }
                }
            }
        }

        if (!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {
            items[i].sellIn = items[i].sellIn - 1;
        }

        if (items[i].sellIn < 0) {
            if (!items[i].name.equals("Aged Brie")) {
```

Scanning

Reading Code Effectively



Scratch Refactoring

Naming as a Process