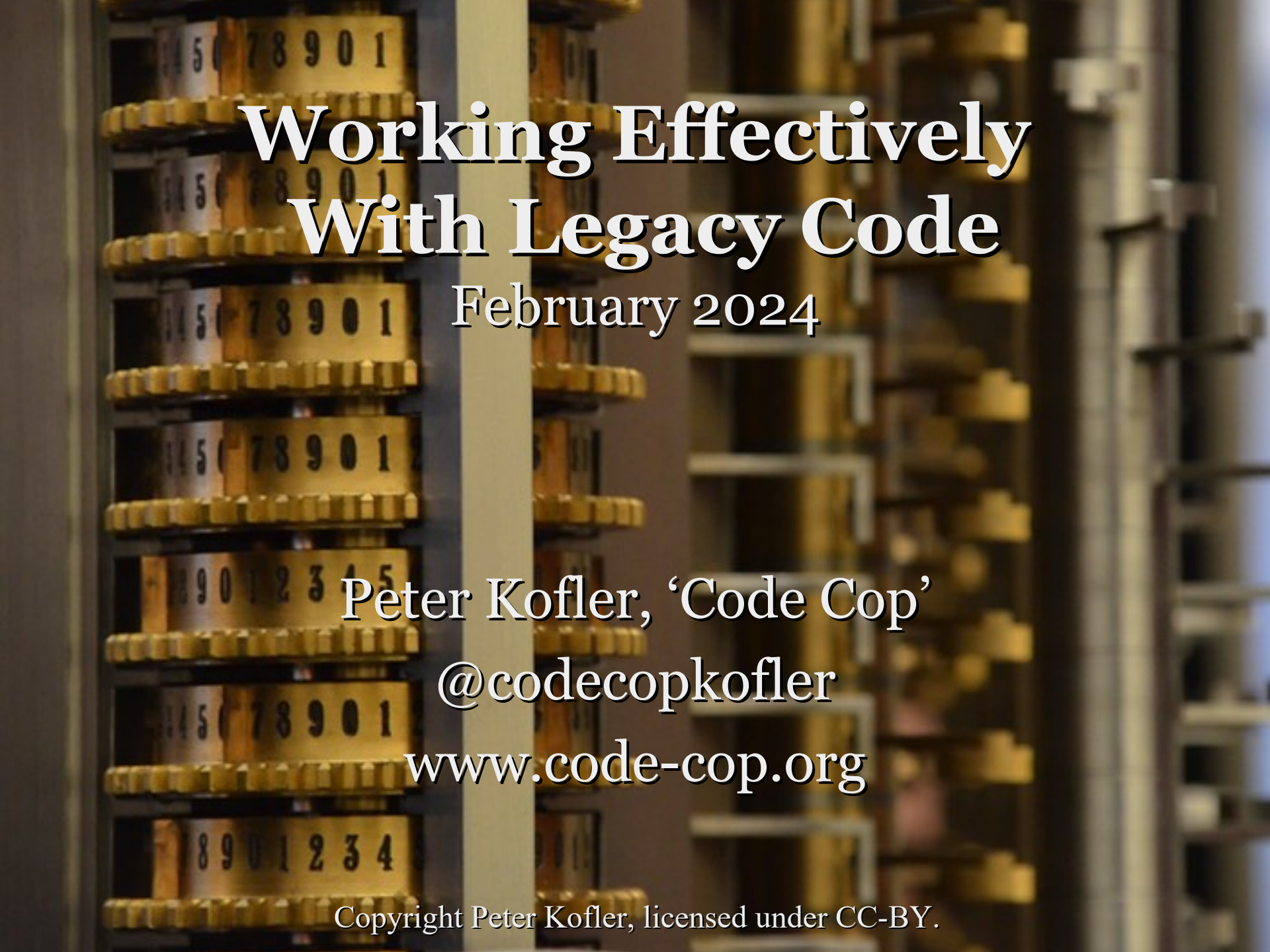


Working Effectively With Legacy Code

February 2024

Peter Kofler, ‘Code Cop’
@codecopkofler
www.code-cop.org

Peter Kofler

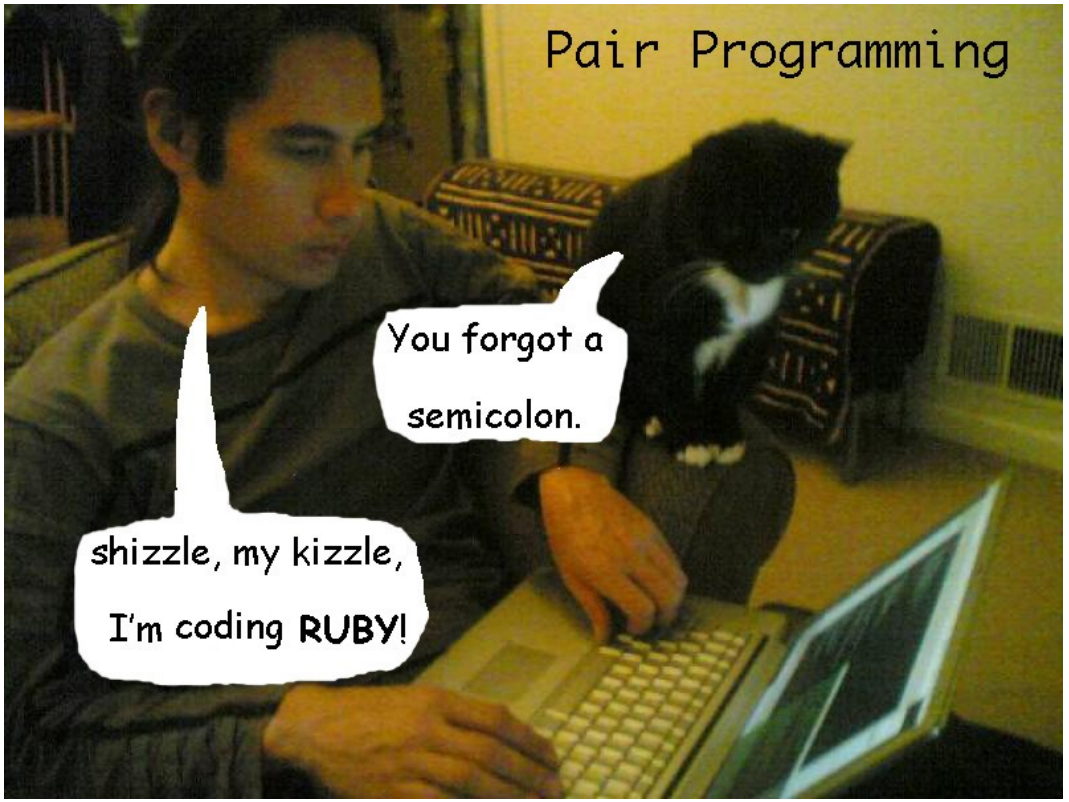
- Ph.D. (Appl. Math.)
- Professional Software Developer for 20+ years
- “fanatic about code quality”
- Independent Code Quality Coach

I help development teams with

- Professionalism
- Quality and Productivity
- Continuous Improvement

Mentoring

- Pair Programming
- Programming Workshops
- Deliberate Practice, e.g. Coding Dojos



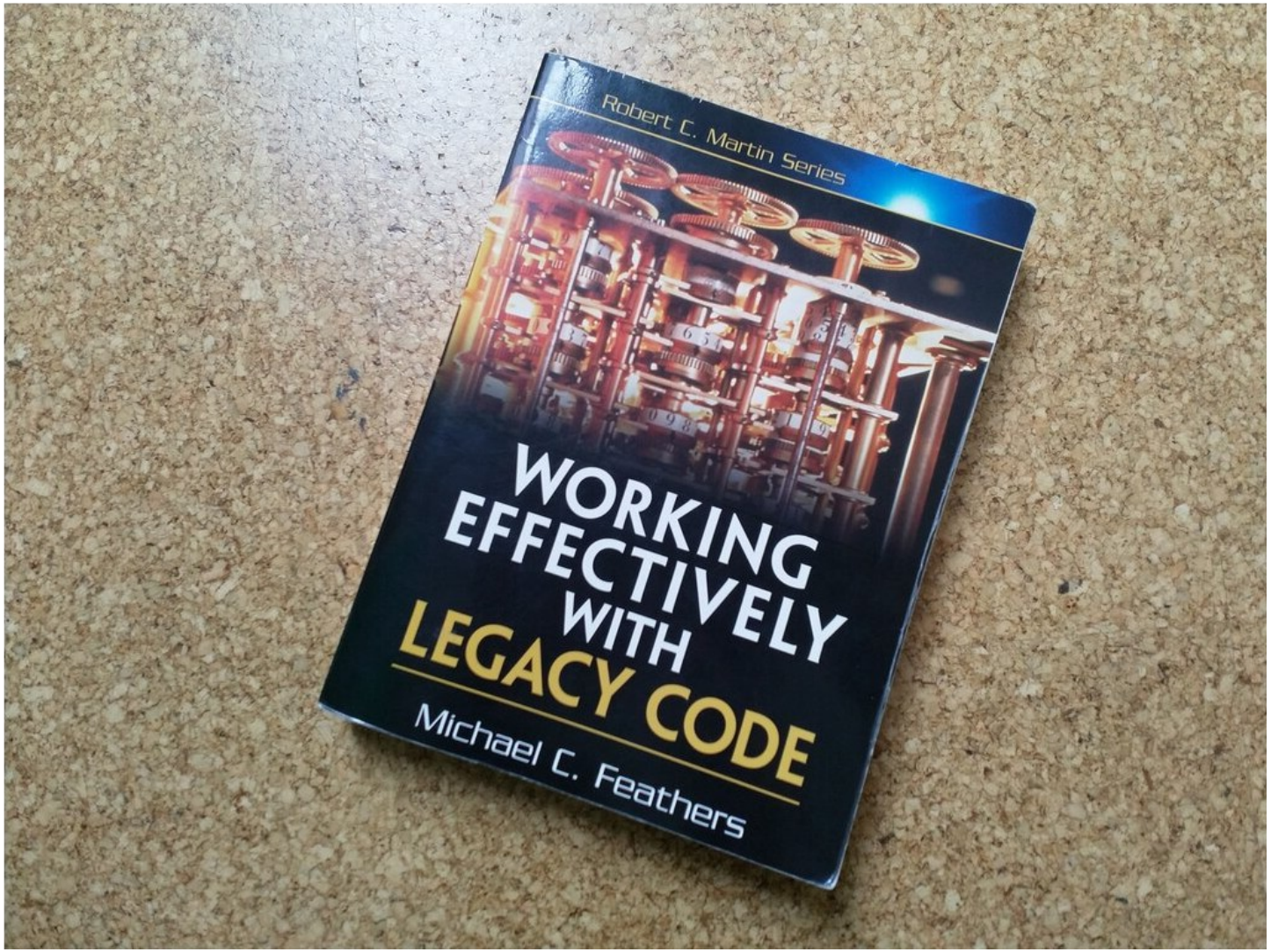
Developing Quality Software Developers

Who is here?

Warm-up

- Find a free frame on the Miro board.
- Write your name on the yellow sticky.
- Fill in the gaps with the most appropriate words from the right.
  (see red arrow)

The Book


Robert C. Martin Series
WORKING EFFECTIVELY WITH LEGACY CODE
Michael C. Feathers

“WELC” / Working Effectively with Legacy Code

- by Michael C. Feathers
- published 2004
- 458 pages
- code in Java, C# and C++, a bit of C
- translated to German, Polish, Russian, Portuguese, Chinese?, Korean

2004 - Is this still relevant?

Totally Relevant!

- Michael Feathers seems underrated because he only wrote this one book.
- The book contains
  - 6 definitions about change points
  - 60 techniques and tricks
- I applied several of these two years ago in a large, real world C++ project and last year in a complex PHP codebase.

What is it about?

Legacy Code is Code without Tests

(Michael Feathers)

The Dilemma

I need to change code to add a test.

I cannot change code because I have no test.

Rule #0 of Surviving Legacy Code: Maximise safety.

(JB Rainsberger)

How do we do it (safely)?

- Make the **smallest** changes to make the code somehow testable.
- Usually separates from evil side effects.
- No breaking changes on any public API.
- Ugly - sometimes makes code worse.
- It’s OK to be bad, it’s only temporary.
- This is the focus of WELC book.

Bring it under Test

- Then create an **integration test** that covers as much of the system as possible.
- Ugly - these are slow tests.
- Ugly - not isolated - not unit tests
- The goal is to detect changes in logic.
- In the end, these test are deleted, because they are bad tests.

Later…

- When we have decent code coverage
- We can improve the code
    - Refactor names and structure
    - Separate responsibilities („units“)
- Then we add nice unit tests
- In the end, the bad tests are deleted.

Demo Time

a.Discount

- Calculates the discount for a purchase in our online shop.
- We want to test it.
- **Problem**: Calls to `MarketingCampaign` are non deterministic.
- We cannot change `MarketingCampaign`.
- We need to change code to enable testing.

Discount Solution

- **Technique**: Parametrise Constructor
- Use constructor chaining or default values to keep original API.
- Use a different `MarketingCampaign` in tests (a test double) to control it.
- This technique even improves design (Open Closed Principle).

b.MarketingCampaign

- controls marketing campaigns of the shop, e.g. when we offer discounts.
- **Problem**: `MarketingCampaign` uses date and time functions.

MarketingCampaign Solution

- **Technique**: Subclass and Override
- Almost no code change, only visibility.
- Use a test specific subclass to override offending methods in tests.
- Bad design: breaks encapsulation.
- Comment/annotate as “visible for test”.

c.Checkout

- Creates the receipt with the calculated tax for a purchase in our online shop.
- **Problem**: Calls to `ReceiptRepository` access the DB which is not available.
- We cannot change `ReceiptRepository`.

Checkout Solution

- **Technique**: Extract and Override Call
- Extract the offending call to a new method. Fully automated refactoring.
- Use a test specific subclass to override offending methods in tests.

d.ShippingCost

- Calculates the shipping cost depending on location and distance.
- **Problem**: A singleton performing slow HTTP calls is used multiple times. The REST server could be offline during tests.
- We cannot change `RestCountriesAPI`.

ShippingCost Solution

- **Technique**: Replace Global Reference with Getter
- Extract the static global data to a new getter method. Fully automated refactoring.
- Use a test specific subclass to override offending methods, return a test double.

e.Checkout

- collects necessary user confirmations during a purchase in our online shop.
- **Problem**: Waits for user input.
- We cannot change other classes…

Checkout Solution

- **Technique**: Extract and Override Factory Method
- Extract constructor with a local factory method. Fully automated refactoring.
- Use a test specific subclass to override offending methods, return a test double.

Conclusion

Conclusion

- List your top three take-away from this presentation and demo.
- What surprised you?
- What is useful for your work?

Peter Kofler

@codecopkofler

www.code-cop.org

CC Images

- Computer History Museum 219 (licensed CC BY-NC by Michael Kappel)
  https://www.flickr.com/photos/m-i-k-e/6706602491/
- Kung Fu Master (licensed CC BY by Sherpas 428)
  https://www.flickr.com/photos/sherpas428/4350620602/
- Pair Programming for Ailurophiles (licensed CC BY-NC-SA by Dav Yaginuma)
  https://www.flickr.com/photos/dav/94735395/
- Stretching (licensed CC BY-NC-ND by Craig A Rodway)
  https://www.flickr.com/photos/m0php/3605168526/
- Jenga (licensed CC BY by Mara Tr.)
  https://www.flickr.com/photos/59145750@N03/5559004171/
- Another Red Light (licensed CC BY-ND by ErWin)
  https://www.flickr.com/photos/lenzmoser/32307397723/
- Dirty hands (licensed CC BY-NC by Nina H)
  https://www.flickr.com/photos/ninahiironniemi/497993647/
- Finish Line (licensed CC BY by studio tdes)
  https://www.flickr.com/photos/thedailyenglishshow/3935979104/
- Wants you (licensed CC BY by Chris Owens)
  http://www.flickr.com/photos/shutter/105497713/