Restructuring Legacy Code: From "Big Ball of Mud" To Domain-Driven Design

NICOLE RAUCH
software development &
development coaching

Workshop Outline

- Presentation of the Problem Domain
- Event Storming of the Problem Domain
- Exploration of the existing codebase
- Presentation of the existing solution
- Presentation of the desired solution
- Guided refactoring towards the desired solution

Our Starting Point

- Business Software
- Very poor code quality
- Planned changes for a module:
  - Bugfixing
  - new features
  - better tests

$\Rightarrow$ A restructuring was required

The Domain

- Financial mathematical software
- Calculate for a bank account for each month:
  - Balance at the last day of the month (ultimo)
  - Average balance of the month

The Domain

- Form small groups (4-6 people)
- Grab some modelling space and some post-its
- Capture Domain Events
  - what happened in the past that was relevant to the business

The Domain

- Form small groups (4-6 people)
- Grab some modelling space and some post-its
- Capture Domain Events
  - what happened in the past that was relevant to the business
- What data is needed to fully determine all aspects of each domain event?

The Code

Find a pair

Check out

`https://github.com/NicoleRauch/RefactoringLegacyCode`

Explore the Code
`00FromPushToPull/src/pushpull/BalancesCalculator.java`

Quick Poll

- Are there quick and obvious improvements?

Quick Poll

- Is the code readable and understandable?

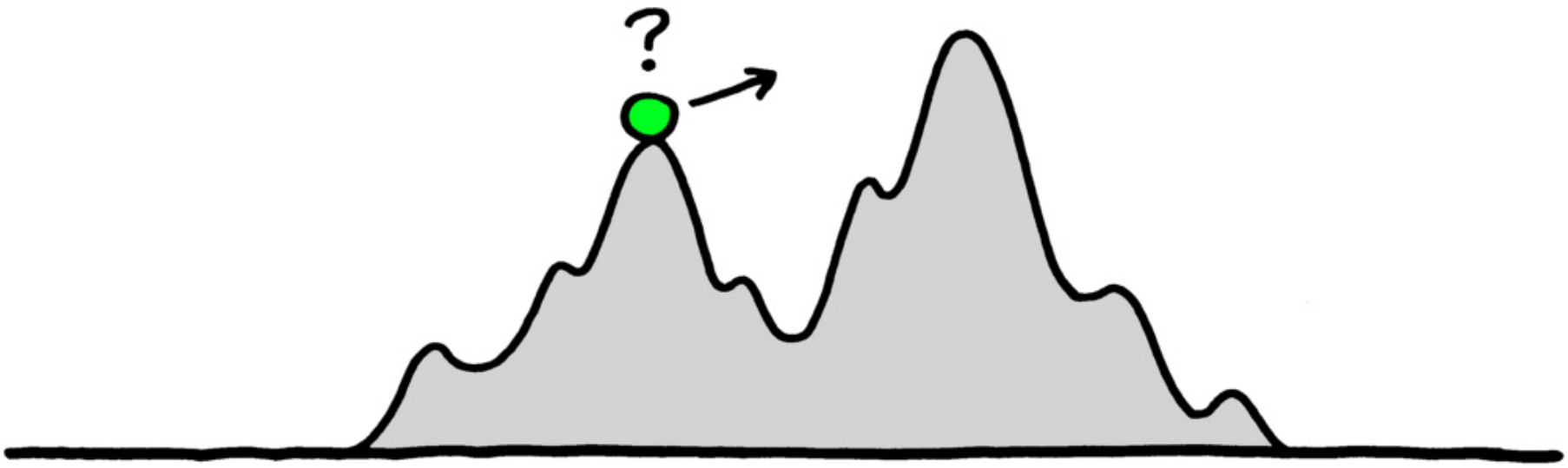
Quick Poll

- Can you explain how it works?

Quick Poll

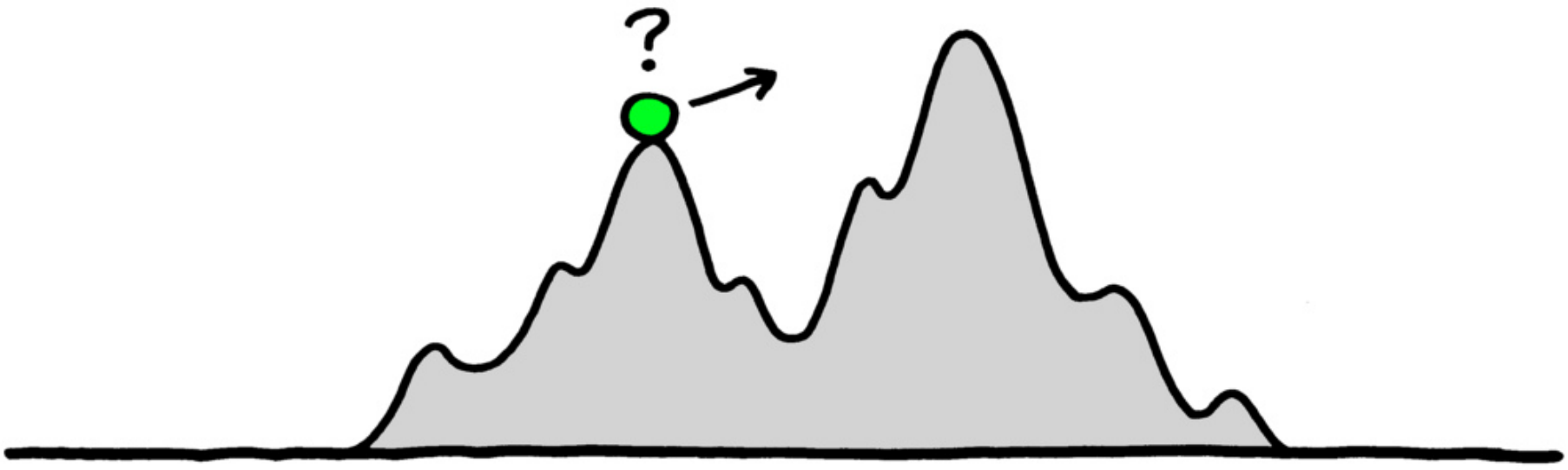
- Can we write tests for individual parts of the code?

Local Optimum



https://www.flickr.com/photos/jurgenappelo/5201851938

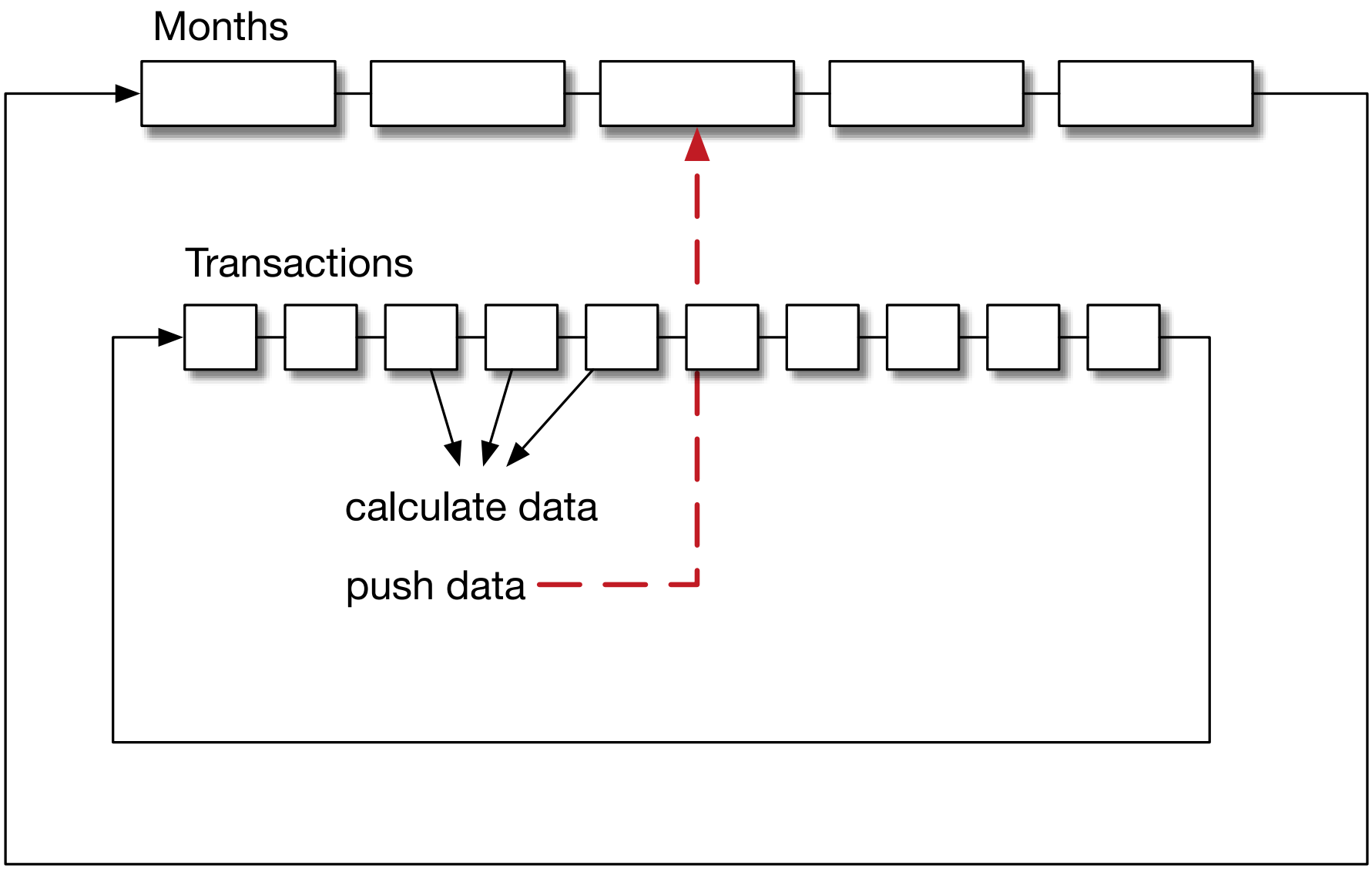
Local Optimum



https://www.flickr.com/photos/jurgenappelo/5201851938

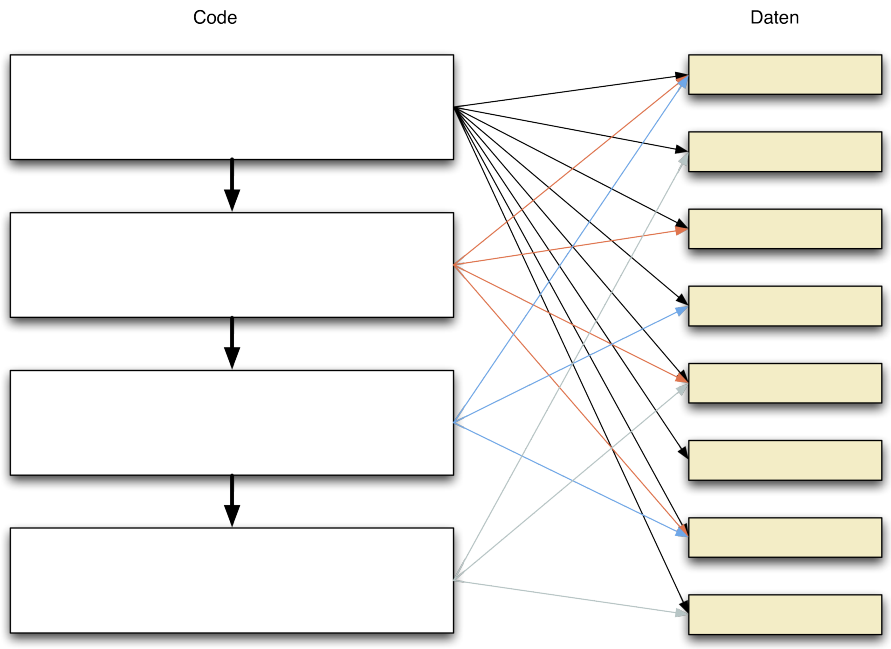
Make it worse to be able to improve!

Code Structure



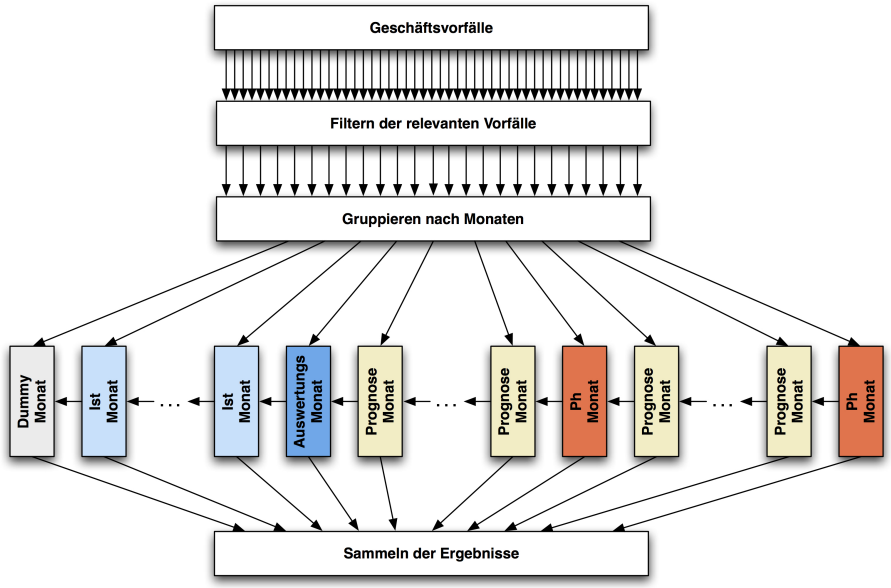
Problems of the Existing Code Structure

- Code writes values into separate data objects („Push“)
- Multiple writing operations for one value
- Parts of the code access previously written values
- Code is driven by the view **from the inside**: What do I need to do in summary to be able to deliver a set of result values?



Goal

- Structure:
  - Code reflects the business logic
- View **from the outside**, driven by the expected results:
  - Which values do I need?
  - How is each value calculated?
  - Which categories of results exist? Similarities, differences?



Overall Approach

- Feature-toggle to compare the old and the new version
  - Identification or creation of a minimal entry point to the restructured area
  - The API of this entry point must remain unchanged
- Important aspects of the restructuring:
  - Driven by business logic
  - Purely structural (NO changes in behaviour)
- Technical goal:
  - Separation of Concerns
  - On-demand-calculation of all values („Pull“)
  - Bonus: Value caching via lazy initialization

Important!

- If in doubt, the existing code shows the correct behaviour!
- Do not change the logic while restructuring!
- Explicit approval of the restructuring
  - It must show identical behaviour (tests, bugs, features)

Workshop Restrictions & Rules

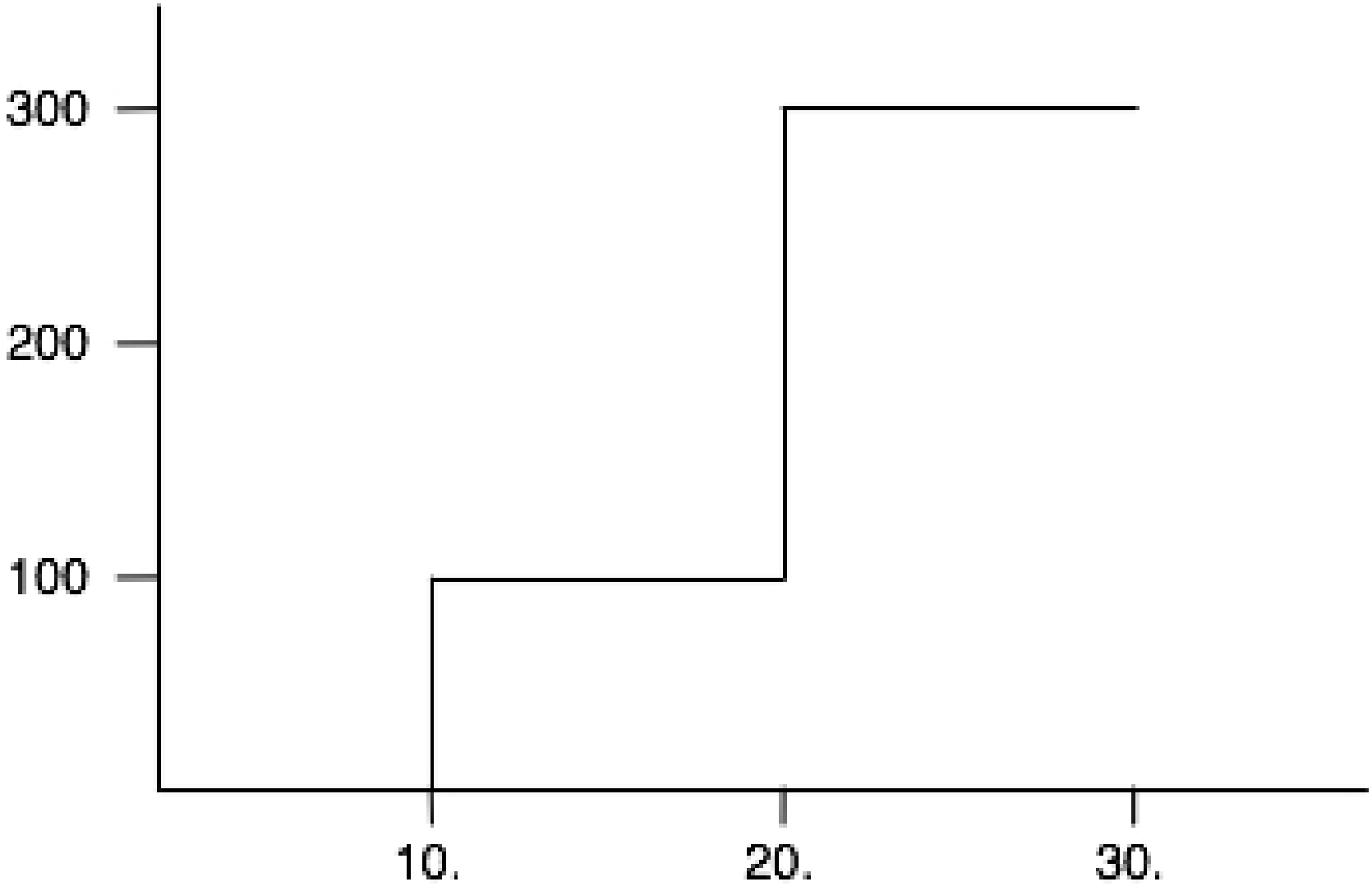
- Change one thing at a time.
- Do not change the outside world (i.e. `..._API` classes), only the internals of our calculator class.
- The code has tests. Run them frequently!

Workshop Structure

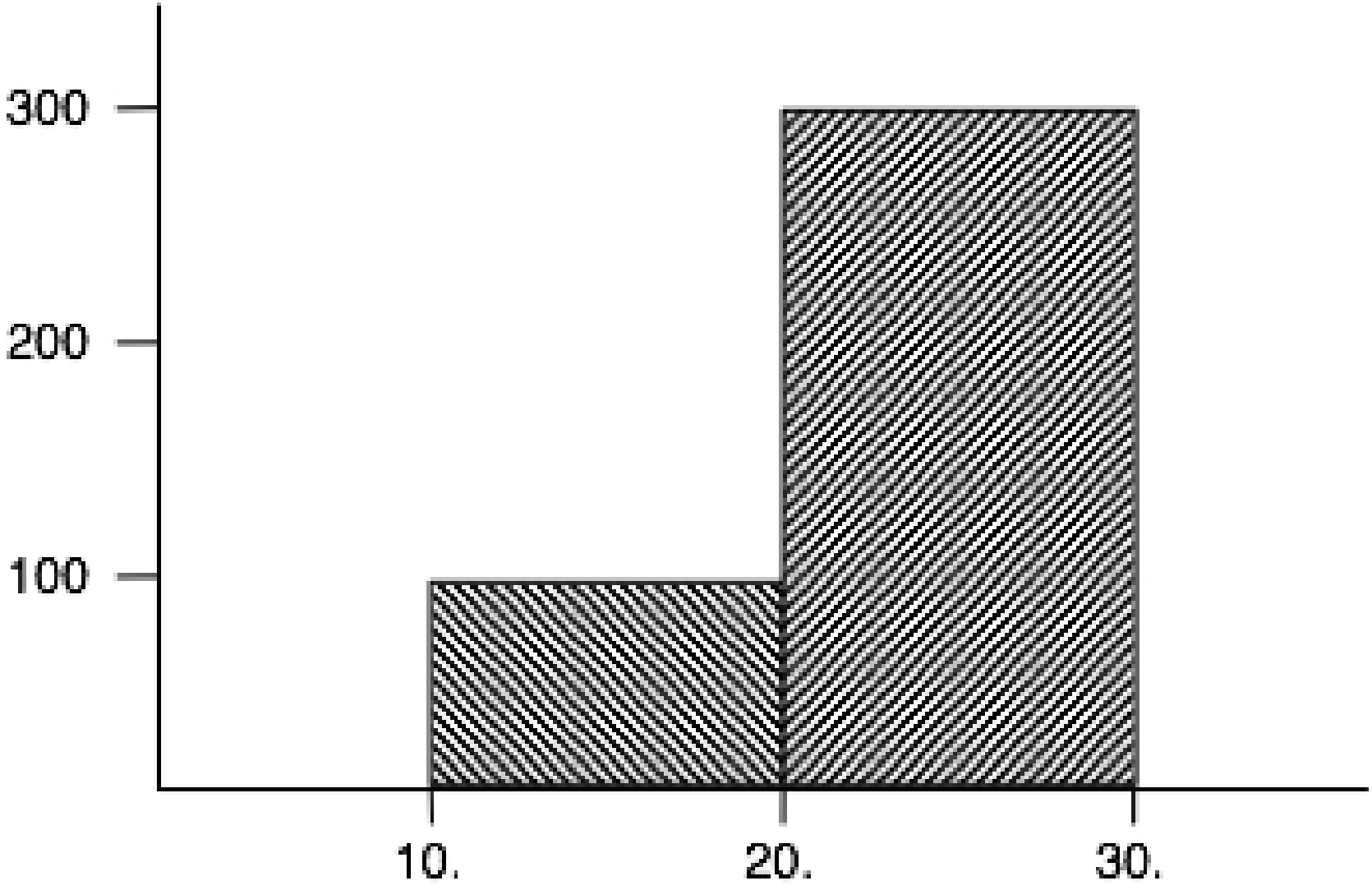
1. Disentangle the for loops, isolate & extract outer loop body
2. One data object per loop iteration
   - no reuse
3. Data object $\rightarrow$ Entity
   - calculate data on demand

Workshop

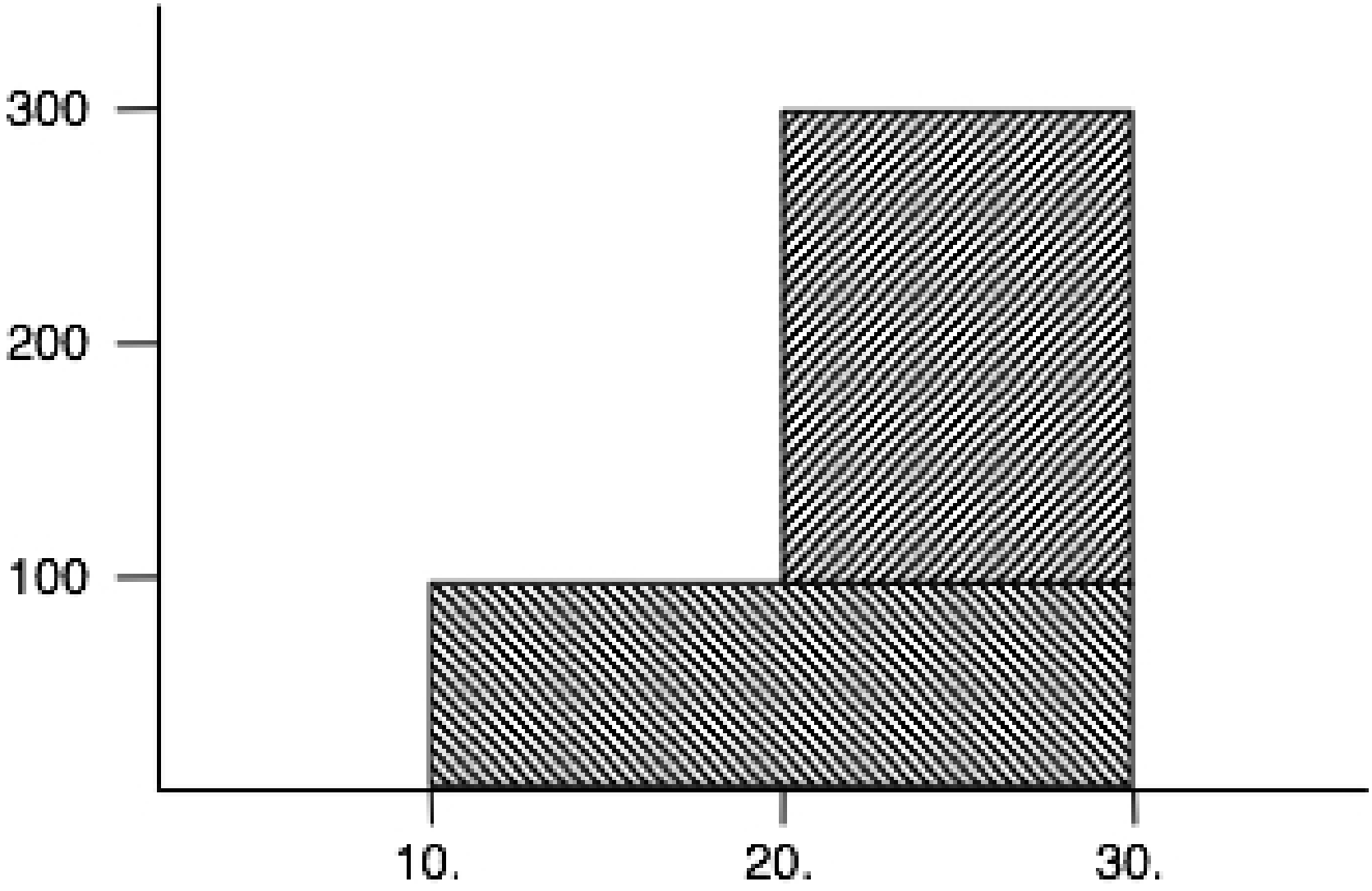
Balance



Initial Average



Final Average



Thank you!

Code & slides at GitHub:

`https://github.com/NicoleRauch/RefactoringLegacyCode`

Nicole Rauch

E-Mail `info@nicole-rauch.de`

Twitter `@NicoleRauch`