msgGillardon

Menschen beraten, *Ideen realisieren.*

Restructuring Code: From "Push" To "Pull"

Andreas Leidig, Nicole Rauch

June 3, 2013

Our Starting Point

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

$\Rightarrow$ A restructuring was required

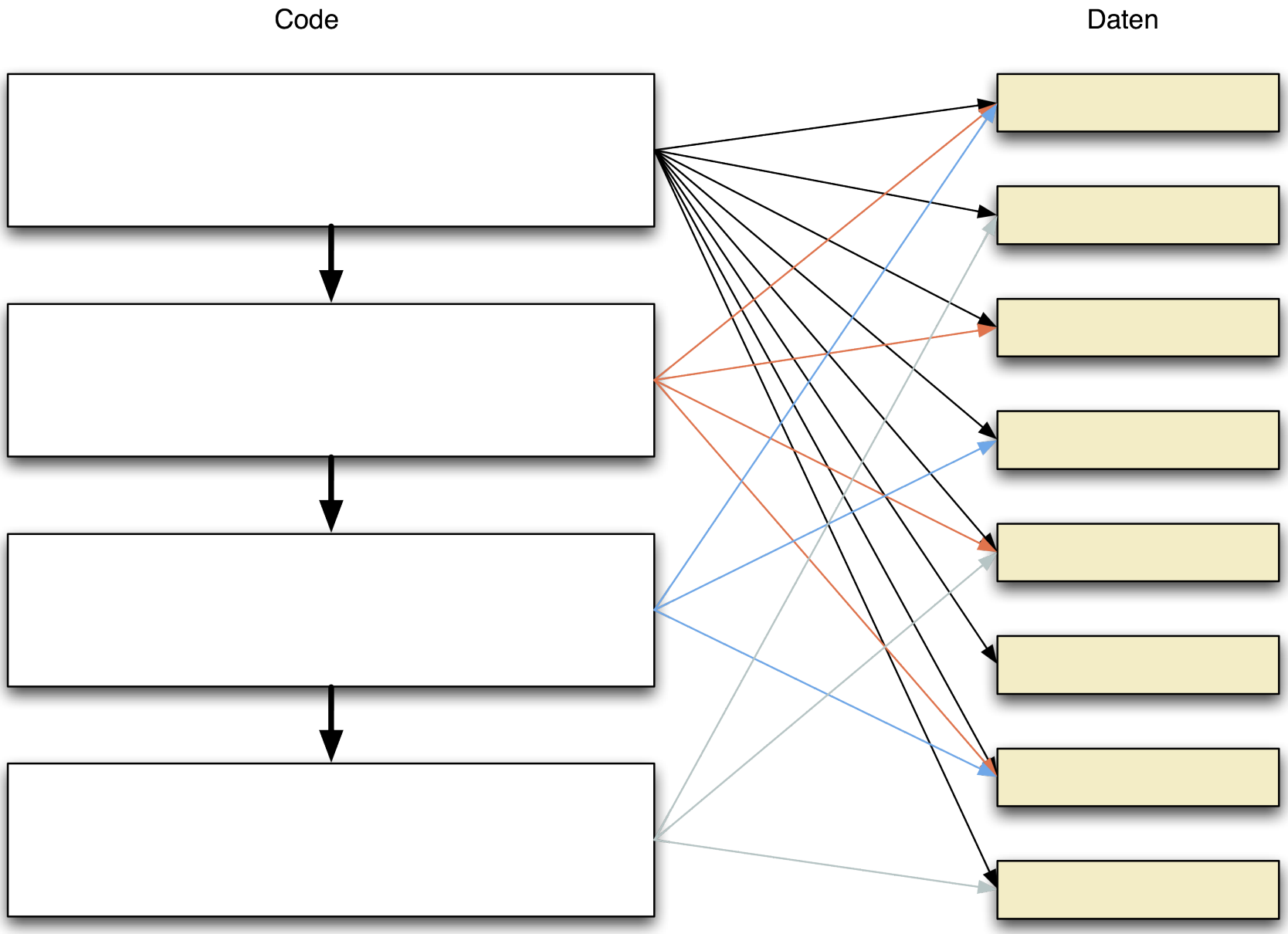
The Domain

- 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

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?

Problems of the Existing Code Structure



Our Approach

- Assumptions:
  - The old code is mostly incorrect and incomplete
  - There is plenty of knowledge regarding the business logic
- Decision:
  - Greenfield reimplementation (with feature-toggle)
  - Creation of new tests in parallel with the new definition of the business logic
  - Continuous adaption of the existing tests

Our Experience

- Problems:
  - It was much more complicated and long-winding to specify the business logic than we had expected
  - Differences compared to the old code are hard to analyze
  - The test coverage is too low
    $\Rightarrow$ We miss relevant cases
- Causes:
  - False assumptions
  - We mixed restructuring and change
  - Arrogance („We know things better than our predecessors“)

$\Rightarrow$ Our restructuring approach was much too naïve

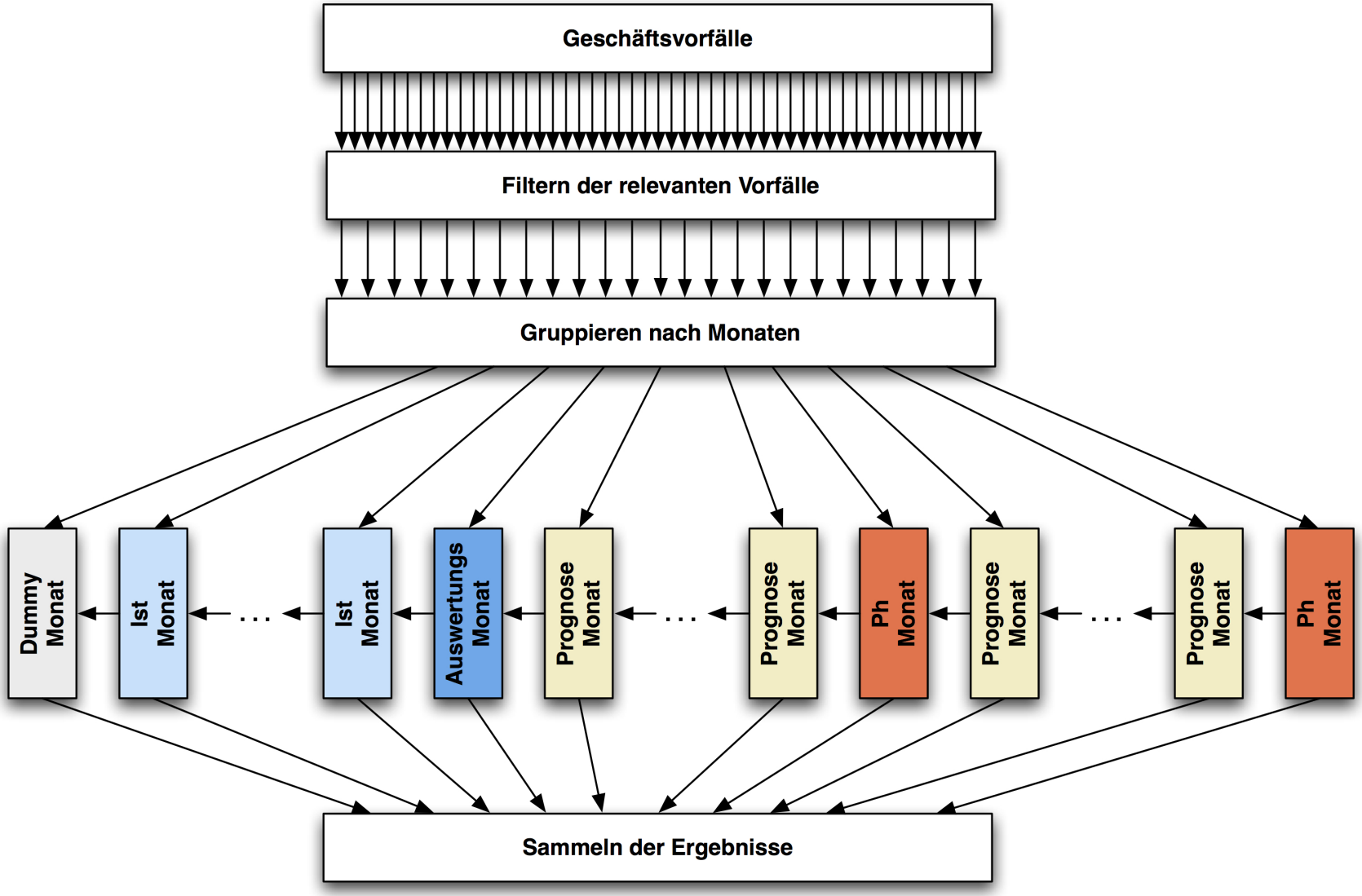
How can this be improved?

- Gather facts in advance (assumptions are not sufficient)
  - How much code coverage do we have?
  - Which business cases are covered?
  - Is there any specification that matches the code?
- 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)

Goal

- Structure:
  - Code mirrors 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?

Goal

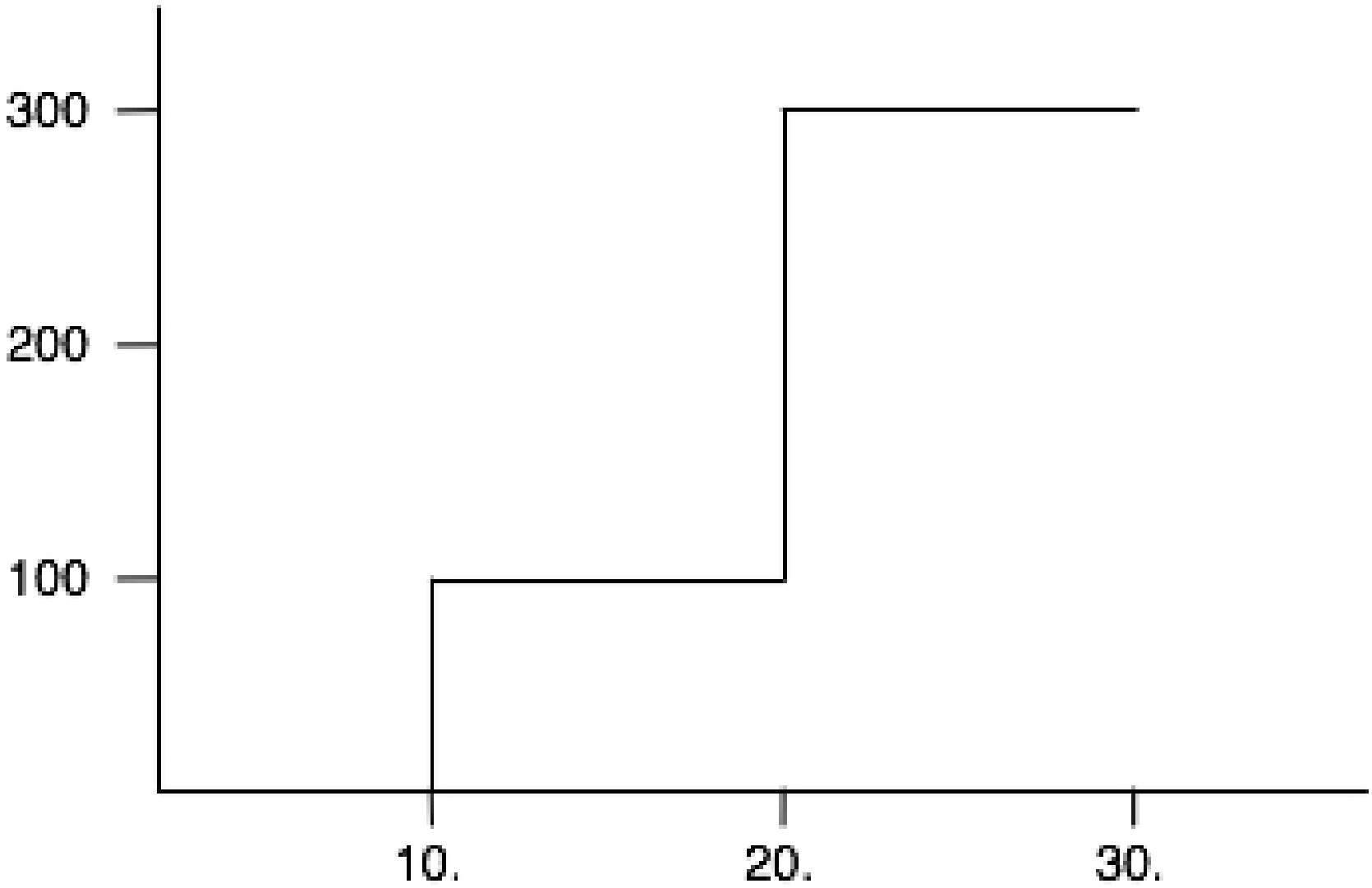

Geschäftsvorfälle
Filtern der relevanten Vorfälle
Gruppieren nach Monaten
Dummy Monat
Ist Monat
Ist Monat
Auswertungs Monat
Prognose Monat
Prognose Monat
Ph Monat
Prognose Monat
Prognose Monat
Ph Monat
Sammeln der Ergebnisse

Ideal Approach

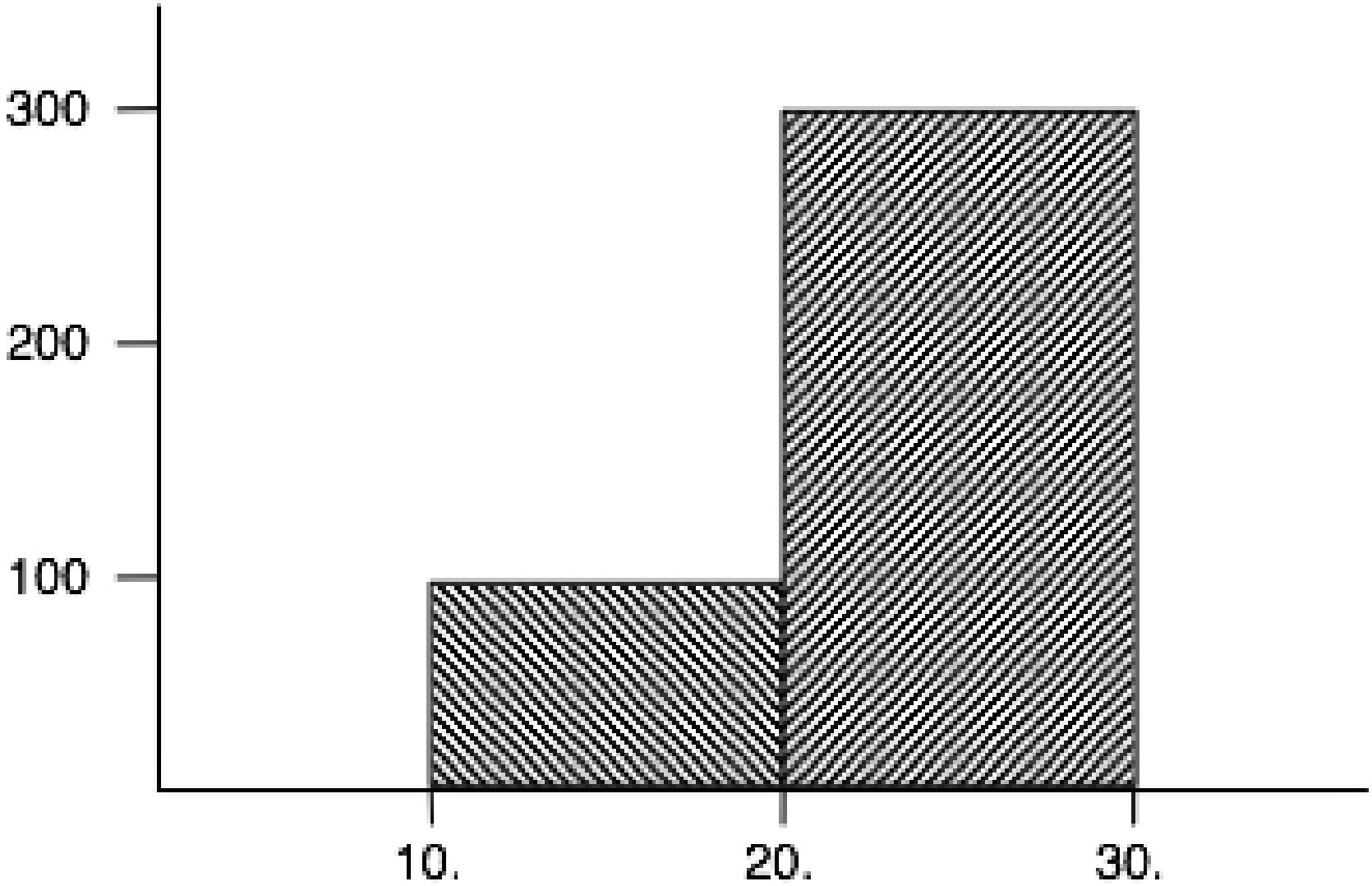
- Feature-toggle to compare the old and the new version
  - Identification or creation of a minimal entry point to the restructured area
  - The API of this entry point must remain unchanged
- Important aspects of the restructuring:
  - Driven by business logic
  - Purely structural
- Technical goal:
  - Separation of Concerns
  - On-demand-calculation of all values („Pull“)
  - Value caching via lazy initialization

Workshop

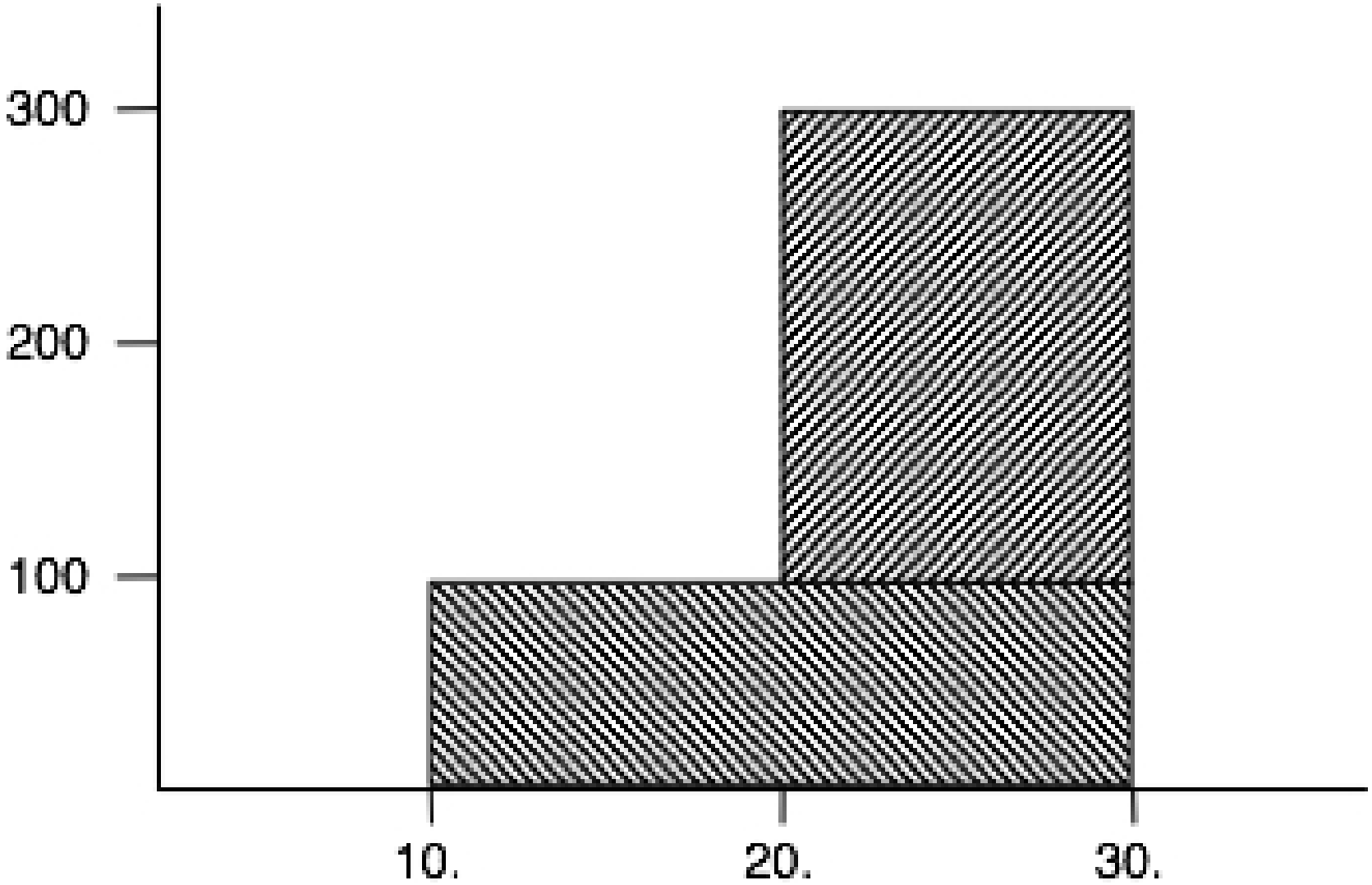
Balance



Initial Average



Final Average



Thank you!

msgGillardon

Code & slides at GitHub:

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

Andreas Leidig

E-Mail `andreas.leidig@msg-gillardon.de`

Twitter `@leiderleider`

Nicole Rauch

E-Mail `nicole.rauch@msg-gillardon.de`

Twitter `@NicoleRauch`