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

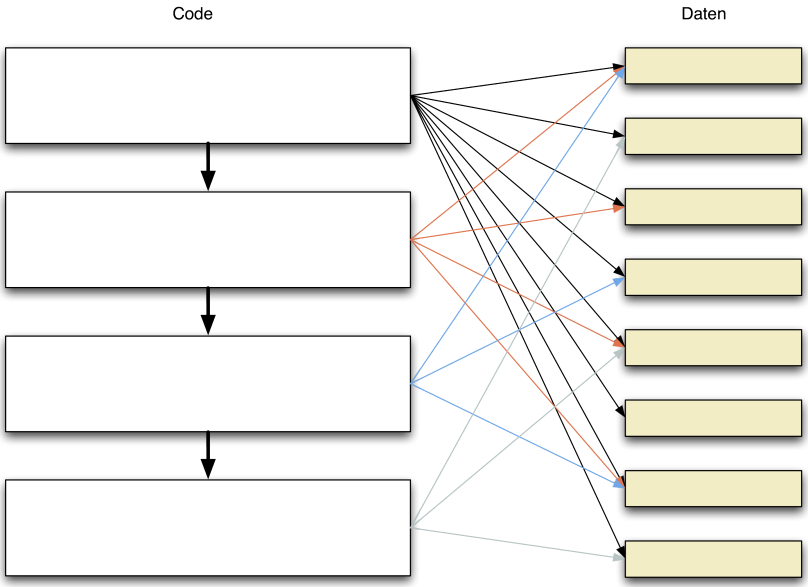
The Domain

- 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?

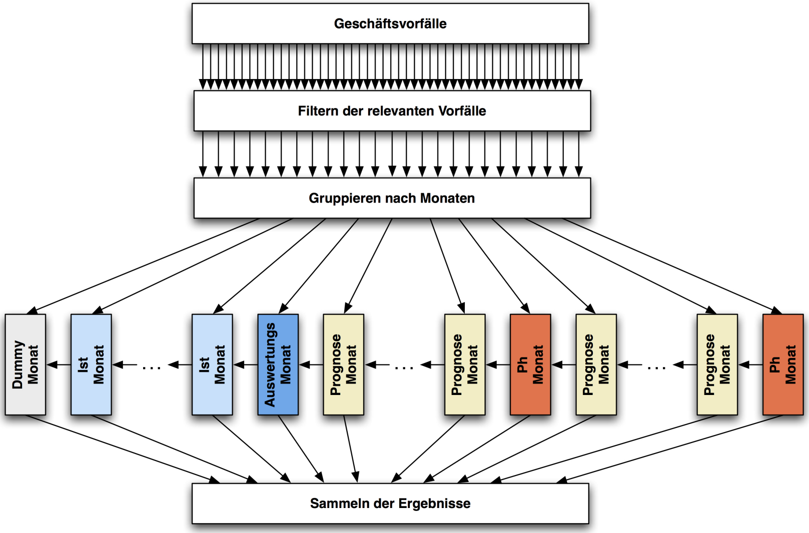
Problems of the Existing Code Structure



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

Good 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“)
  - 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)

Pattern Application - Code Structure

The pattern is applicable if the code structure is similar to this:

- a sequence of input data elements
- very few classes with logic
- a sequence of output data elements
- output sequence is constructed early in the program run
- output sequence is filled during program run

Pattern Application - Code Workflow

The basic mechanics of the logic class(es) is as follows:

- initially, the sequence of output data is not filled
- the main logic part consists of two interlaced loops:
  - the outer loop iterates over the sequence of output data objects
  - the inner loop iterates over the sequence of input data objects and calculates the data for one output data object
  - at the end of the inner loop, the calculated data is written to that output data object ("push")
- finally, the sequence of output data is filled
- the logic class(es) may contain multiple of these interlaced loops in sequence, where a subsequent loop may access and modify the output data that a previous loop has calculated and pushed to the sequence of output data

Pattern Application

The pattern can be broken up into three major parts:

1. Disentangle the for loops, isolate the loop body and extract it
2. Build one data object for each loop iteration (instead of reusing the same object over and over again)
3. Isolate the calculations of the distinct values into separate methods and calculate them on demand

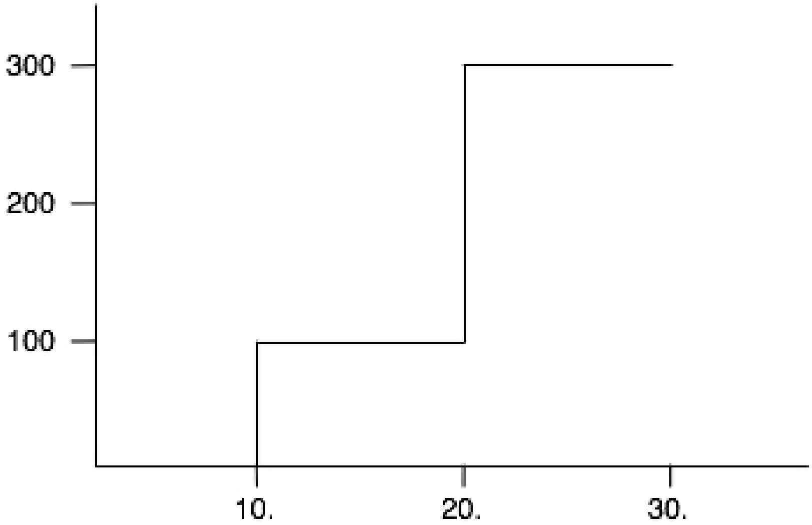
Workshop Restrictions & Rules

- Change one thing at a time.
- While applying this pattern, we neither want to change the outside world nor our calculator class API, only the internals of our calculator class.
- The result objects represent the outside world in our example. Therefore, we will leave the result objects and the signature of the calculator class API untouched.

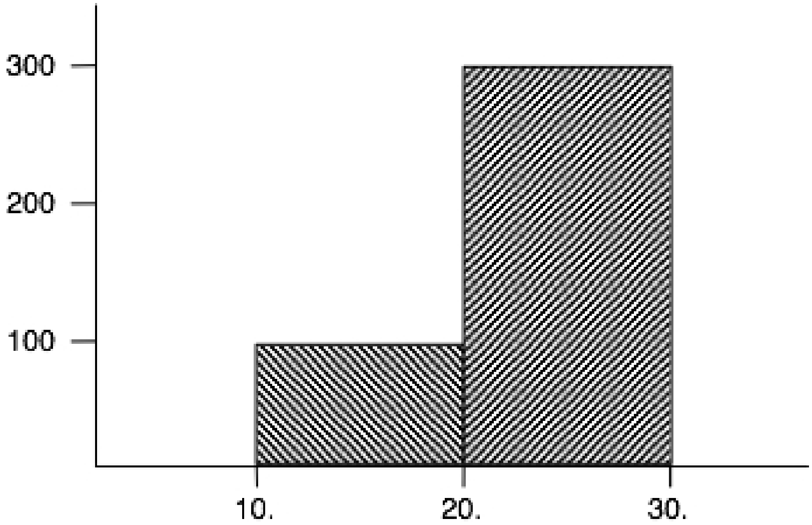
Workshop

Workshop

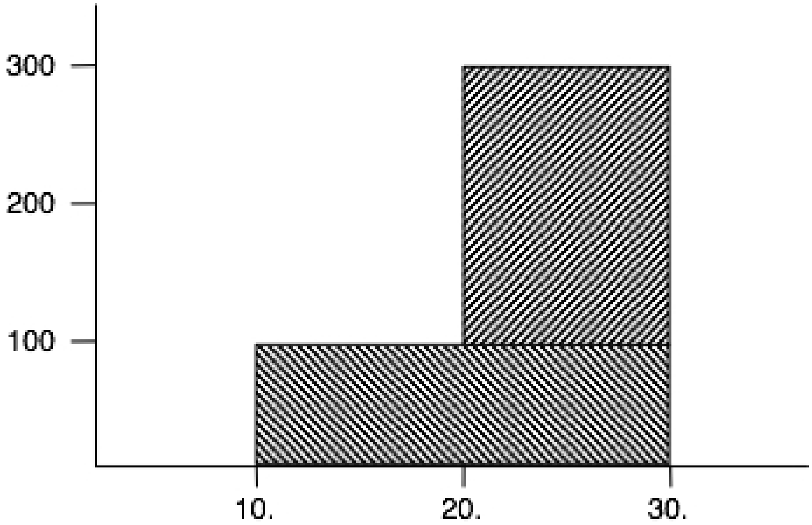
Balance



Initial Average



Final Average



Thank you!

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`