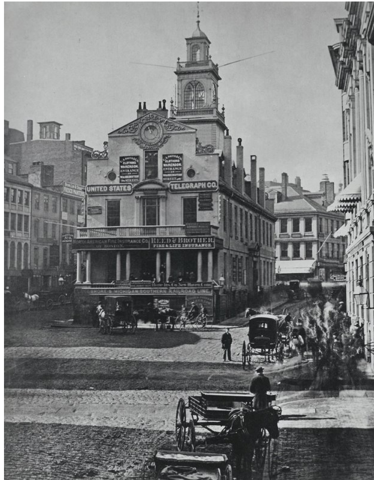
msgGillardon
Menschen beraten, *Ideen realisieren.*

Refactoring von Legacy Code

Andreas Leidig, Nicole Rauch

10. Mai 2012

Legacy – Vermächtnis



http://www.flickr.com/photos/bostonlandmarkscommission/

http://www.flickr.com/photos/josepha/

Legacy – Vermächtnis

http://www.flickr.com/photos/venana/

Legacy – Vermächtnis

http://www.flickr.com/photos/theo_reth/

Legacy Code – Was ist das eigentlich?

- Michael Feathers: „Code without tests"
  - „With tests, we can change the behavior of our code quickly and verifiably."
  - „Without them, we really don't know if our code is getting better or worse."

Legacy Code – Was ist das eigentlich?

- Legacy Code ist
  - von anderen
  - alt
  - unübersichtlich
  - buggy
  - mühsam zu erweitern
  - ...

Unser Ausgangspunkt

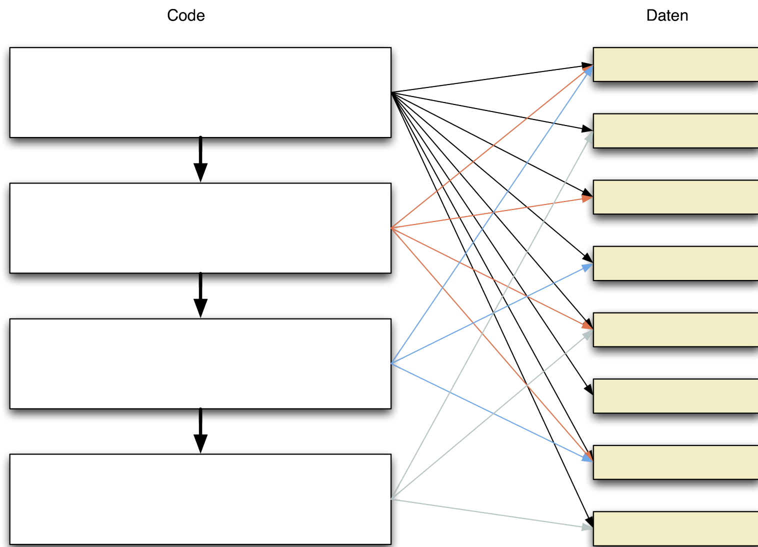
- Betriebswirtschaftliche Software
- Extrem schlechte Codequalität
- Anstehende Veränderungen für ein Modul:
  - Bugfixing
  - neue Features
  - bessere Tests

⇒ Ein Umbau war notwendig

Die Domäne

- Finanzmathematische Software
- Für ein Konto monatsweise ausgeben:
  - Kontostand am Monatsletzten
  - Durchschnittlicher Kontostand im Monat

Vorhandene Codestruktur



Probleme der vorhandenen Codestruktur

- Code schreibt Werte in separate Datenobjekte („Push“)
- Mehrfache Schreibzugriffe auf denselben Wert
- Codeteile greifen auf vorher geschriebene Werte zu
- Code ist getrieben vom Blick von innen: Was muss ich alles in Summe tun, um eine Menge von Ergebnissen abliefern zu können?

Unser Ansatz

- Annahmen:
  - Der alte Code ist großteils inkorrekt und lückenhaft
  - Es existiert umfassendes Wissen über die gewünschte Fachlogik
- Entschluss:
  - Neuimplementierung auf der grünen Wiese (mit Feature-Toggle)
  - Erstellung neuer Tests anhand der parallel zu definierenden Fachlichkeit
  - Kontinuierliche Anpassung der vorhandenen Tests

Unsere Erfahrungen

- Probleme:
  - Die Ausarbeitung der fachlichen Spezifikation ist viel komplizierter und aufwändiger als gedacht
  - Abweichungen zum alten Code sind schwer zu analysieren
  - Die Testabdeckung ist zu gering
$\Rightarrow$ Wir übersehen relevante Fälle
- Ursachen:
  - Falsche Annahmen
  - Vermischung von Umbau und Änderung
  - Arroganz („Wir wissen es besser als unsere Vorgänger“)

$\Rightarrow$ Wir sind den Umbau viel zu naiv angegangen

Wie geht es besser?

- Vor dem Umbau Fakten sammeln, Annahmen allein genügen nicht
  - Wie hoch ist die Testabdeckung? (Coverage)
  - Welche fachlichen Fälle sind abgedeckt?
  - Existiert eine zum Code passende Spezifikation?
- Grundsätzlich gilt: Im Zweifel hat der vorhandene Code Recht!
- Keine Änderungen an der Logik während des strukturellen Umbaus!
- Explizite Abnahme des Umbaus
  - Das Verhalten muss exakt gleich sein (Tests, Bugs, Features)

Ziel

- Struktur:
  - Spiegelbild der Fachlichkeit
- Blick von außen aus Sicht der Ergebnisse:
  - Welche Werte will ich haben?
  - Wie berechnet sich welcher Wert?
  - Welche Kategorien von Ergebniswerten gibt es? Gemeinsamkeiten, Unterschiede?

Ideale Vorgehensweise

- Feature-Toggle zum Vergleichen von alter und neuer Version
  - Identifikation bzw. Schaffen eines minimalen Eingriffspunkts
  - Beibehalten der externen API an diesem Eingriffspunkt
- Wichtige Aspekte des Umbaus:
  - Fachlich motiviert
  - Rein strukturell
- Technisches Ziel:
  - Separation of Concerns
  - On-Demand-Ermittlung aller Werte („Pull“)
  - Werte-Caching mittels Lazy-Initialization

Umbau: Vorbereitung

- Existierenden Code duplizieren
- Feature Toggle an den Aufrufstellen einbauen

Vorarbeiten

- Java-Datumsarithmetik kapseln, z. B.
  - Joda Time
  - Eigene Klassen
- Namensgebung von Variablen verbessern

Strukturellen Code isolieren

- Schleifen aufräumen, vereinheitlichen und zusammenfassen
- Dazu bei Bedarf Methoden inlinen

Aspekte isolieren

- Werteobjekt einführen, das alle relevanten Ergebnisse des Schleifenrumpfs zusammenfasst
- Schleifenrumpf in Methode extrahieren
- Methode in das Werteobjekt verschieben

Push in Pull umwandeln

- Pro Ergebniswert ein Duplikat dieser Methode
- Jeweils Irrelevantes aus den Methoden entfernen
- Die Methoden zum Berechnen der Ergebniswerte in Getter umwandeln
- Parallel dazu Unit-Tests aufbauen

Vervollständigung

- Refactoring:
  - Extrahieren von Methoden
  - Zusammenfassen gleicher Funktionalität
  - Generelle Aufräumarbeiten
- Danach:
  - Technische Bereinigung von Algorithmen, fachliche Korrektur der Berechnungslogik

Vielen Dank!

msgGillardon

Code & Folien auf 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`