Von Legacy Nach Symfony

Refactoring PHP

TNG TECHNOLOGY CONSULTING

München, 14.6.2013
Stefan Kirsch

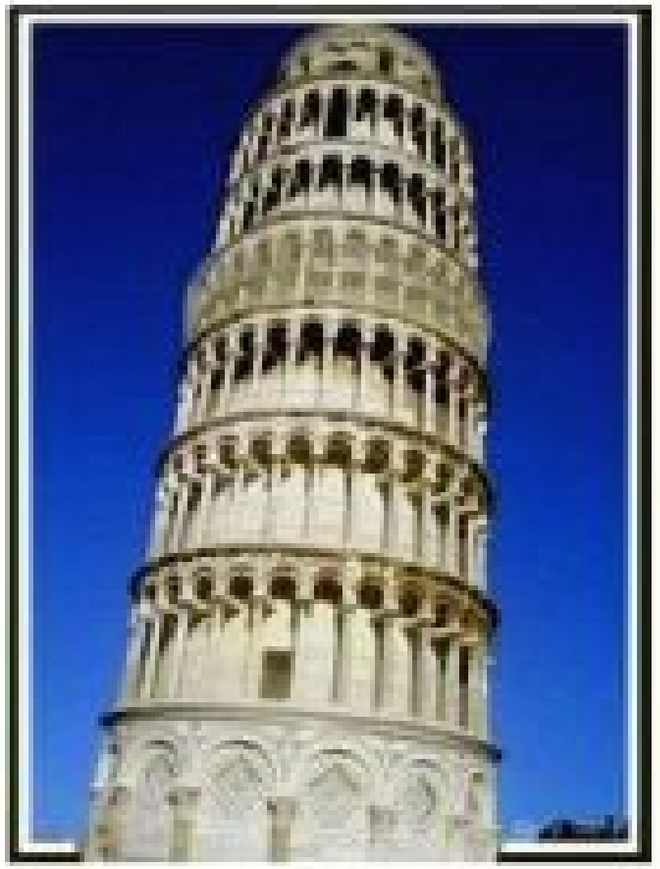
Agenda

- Legacy Code
- Das Projekt
- Refactoring und Testing
- Rewrite
- Die Strategie

Legacy Code

Legacy Structure

- Struktur
  - Patterns
  - Schichten
  - Interfaces
  - Reusability



Legacy Bugs

- Hohe Buganzahl
  - Entwicklungsprozess
  - Feature > Bug Fix
- Tests
  - Code nicht testbar
  - Keine automatisierten Tests

"Legacy code is code without tests."

- Michael Feathers

Das Projekt

MANOffice®

- Redaktionssystem zur Erstellung von Fahrzeugdokumentation
- Rewrite des bestehenden Systems: 3.0
- Übernommen von bisherigem Entwickler
- Hochkomplexe Anforderungen
- Nicht feature complete und instabil

Architektur und Technik

- Backend: PHP 5.2/5.3
  - Selbstentwickeltes Web Framework und Template Engine
  - Stark Event-basiert
- Frontend: ExtJS 3
- Datenbank: Oracle 10g

Probleme

- Hohe Anzahl Bugs
- Schwer verständliche, veraltete und falsch implementierte Patterns
- Event-Architektur führt zu vielen Seiteneffekten
- Schichtenmodell in großen Teilen verletzt
- Keine Tests

Der Auftrag: Refactor

Das Ziel

- Betriebssicherheit
- Performance
- Feature Completeness
- Effiziente Weiterentwicklung

"Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior."

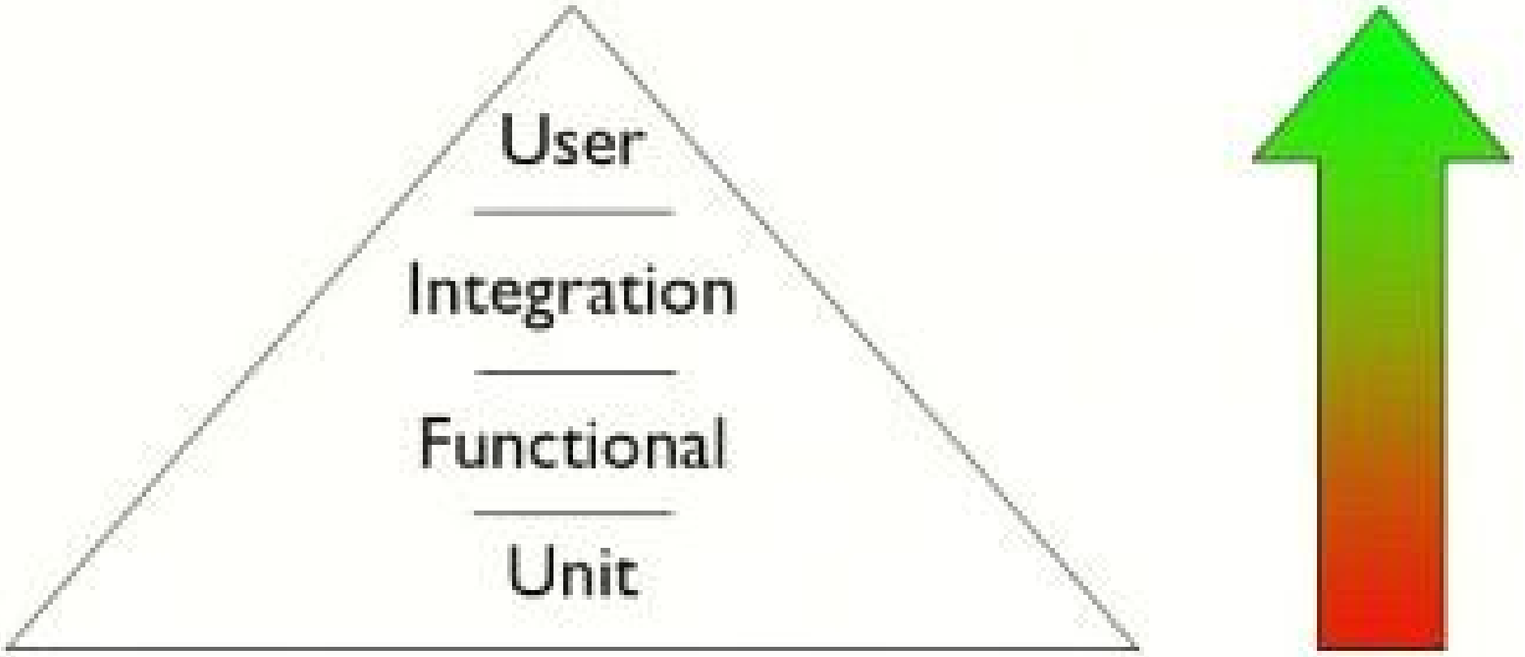
- Martin Fowler

Refactoring

- "External Behaviour" muss klar definiert sein
- "Internal Structure" umfasst nicht die generelle Architektur
- Keine Tests = Kein Refactoring

Testing

Testing



+ Automatisierung

Continuous Integration

Die Grenzen von Refactoring

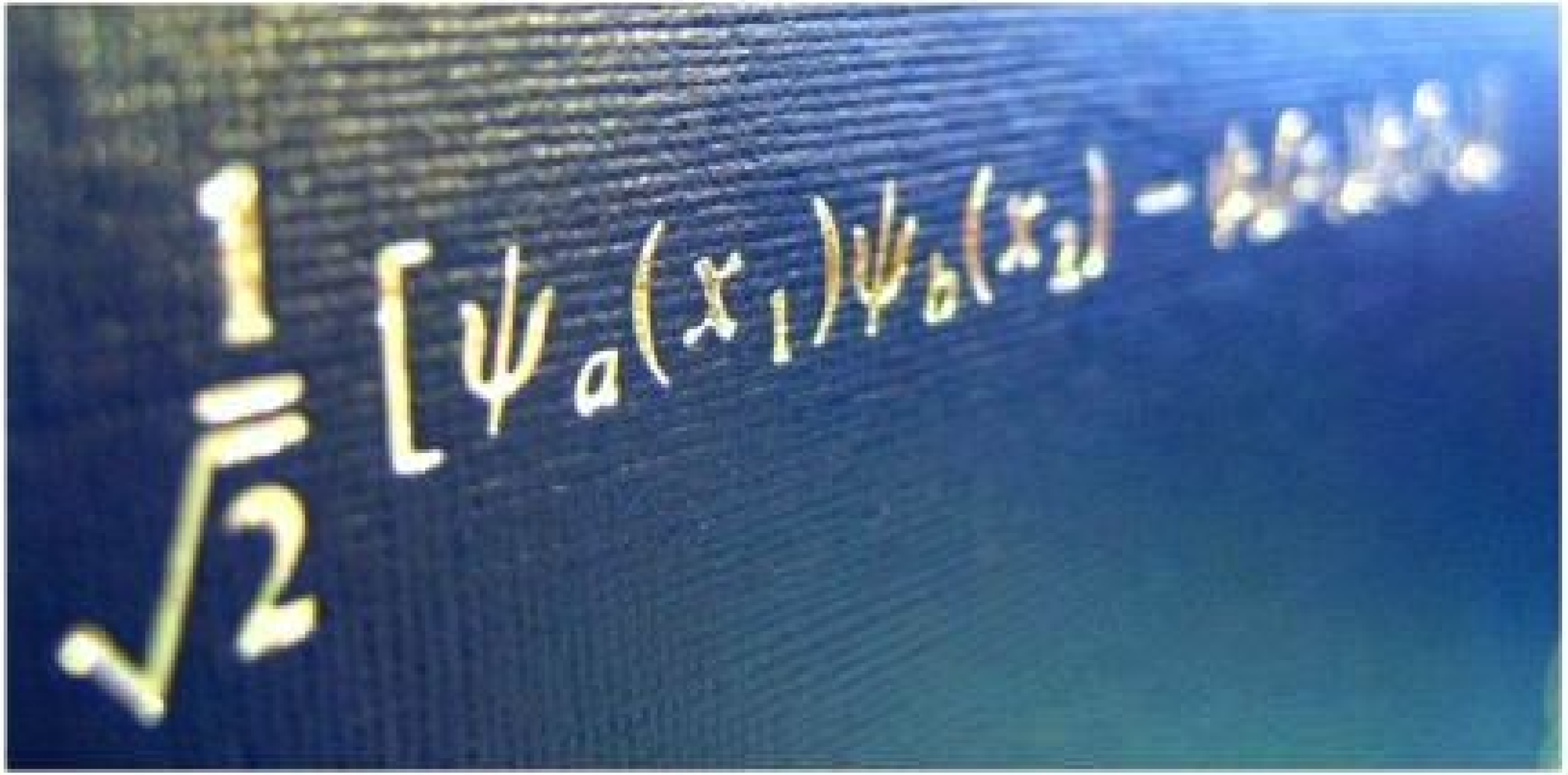
- System
  - ... untestbar
  - ... architekturell ungeeignet
  - ... hat zu viele technische Schulden
- Echtes Refactoring verursacht dann höheren Aufwand als Rewrite

... Rewrite?

Wenn Refactoring nicht ausreicht

Rewrite ?!

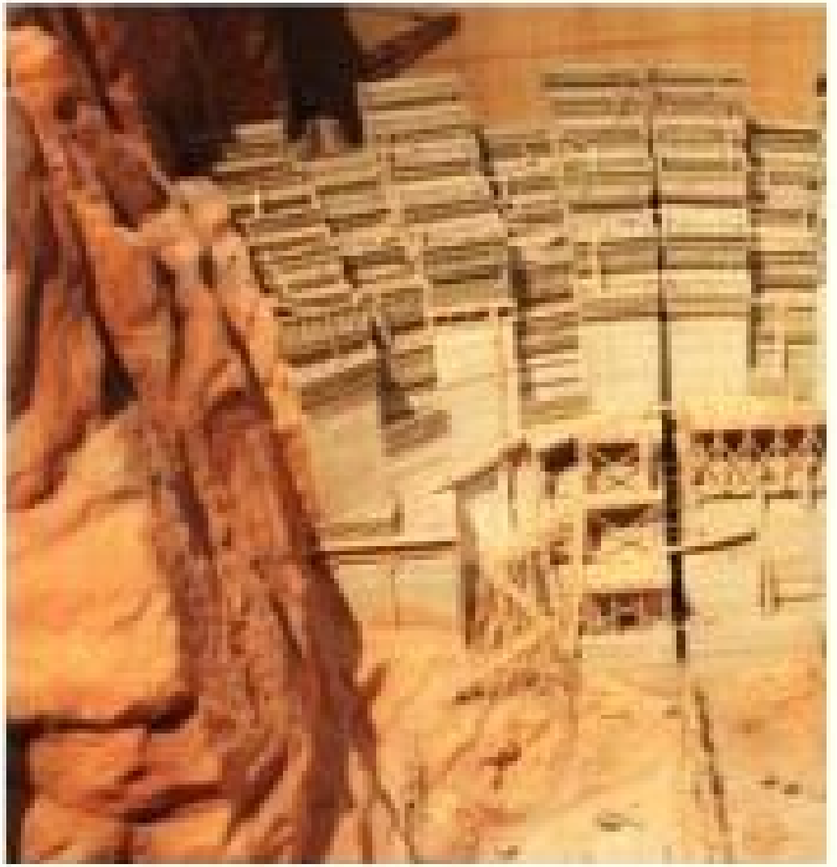
- "Der Aufwand wird explodieren!"
- "Wir wissen nicht, ob das Ergebnis besser wird als das was wir haben."
- "Wir brauchen aber neue Features - jetzt."
- "Wir sehen zu lange keinen Fortschritt."



Antworten

1. Aufwand

- Echtes Refactoring erfordert eine umfangreiche Test Suite
- Kann nur iterativ erstellt werden
- Großflächiges Umbauen des Codes + Refactoring der Test Suite
- Ein System ohne Tests ist per Definition wertlos

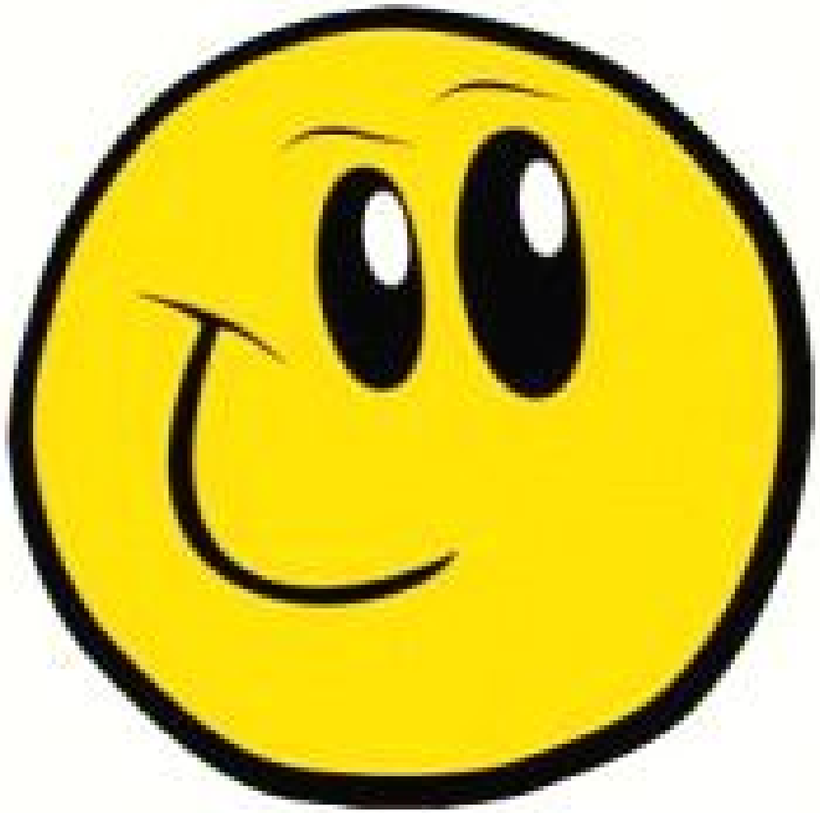


Prozess

- Scrum
- XP
- CI
- Strikte DoD

2. Ergebnis

- Nutzung von Frameworks und fertigen Komponenten
- Voll getestet = bug free
- Requirements liegen bereits in Form des bestehenden Systems vor



Symfony2

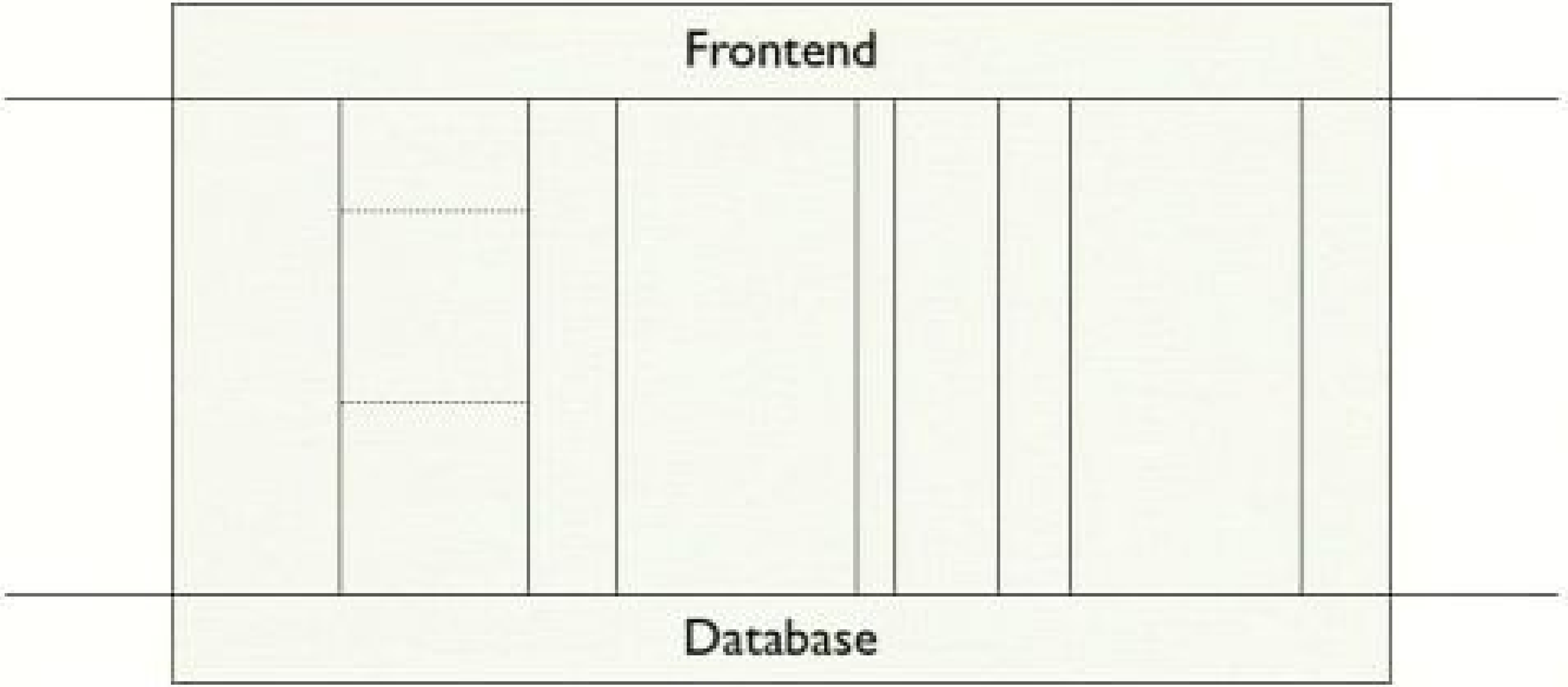
- Modulares Web Framework
- Erprobte Patterns, z.B.
  - Dependency Injection
  - Annotations
- Community

Symfony

3. Features

- Planung des Rewrite in sinnvollen Schnitten
- Nutzung von existierenden Interfaces
  - Frontend - Backend - Datenbank
  - Trade Offs erforderlich
- Rewrite entlang der Feature Roadmap
- Beschränkung auf funktionale Einheiten

Schnitte



4. Fortschritt

- Keine "Schwarzes Loch"-Situation
- Proxy-Setup:
  - Alt- und Neusystem laufen parallel
  - Nicht migrierte Features werden von Altsystem bedient
- Jederzeit deploybares System

PHP

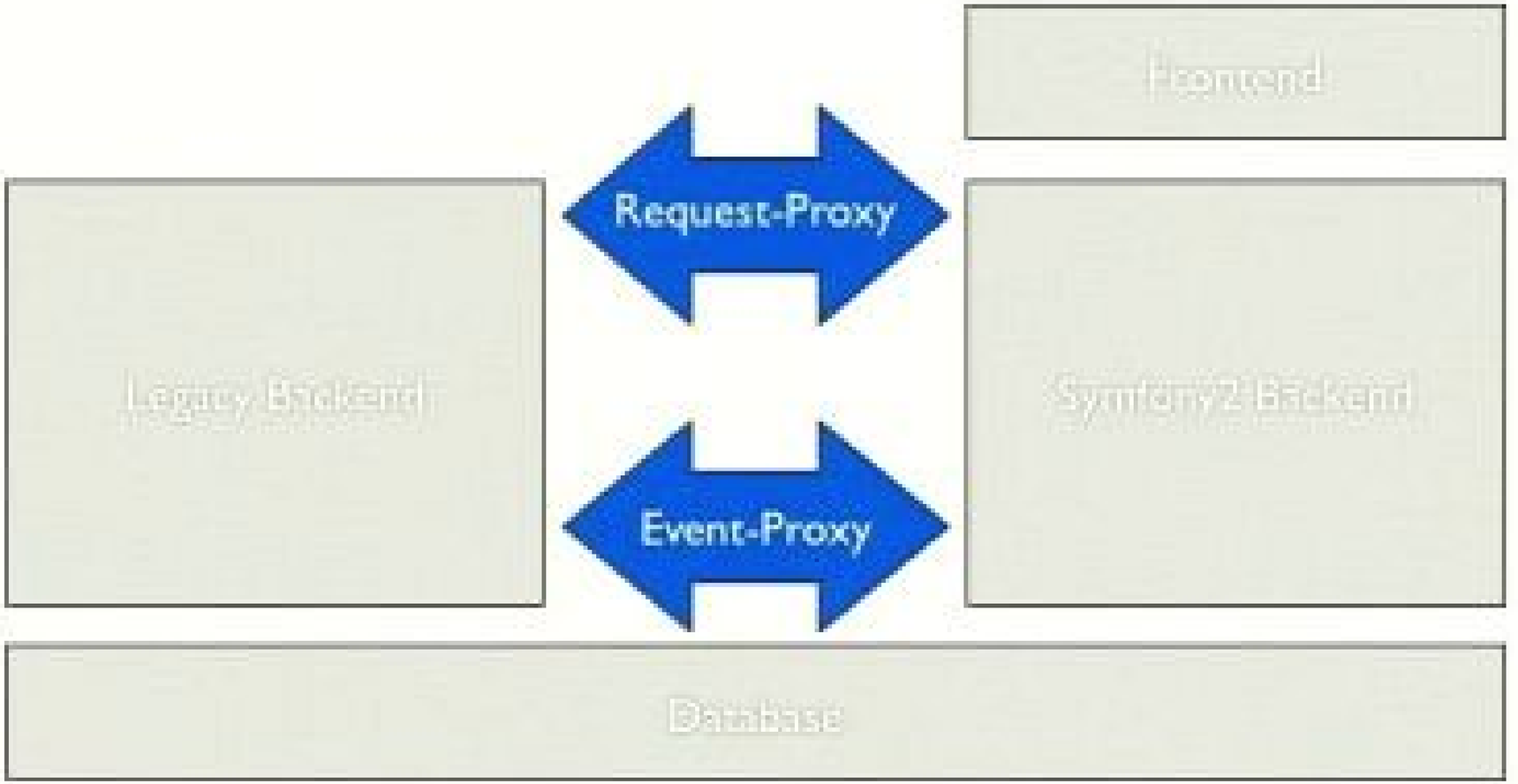
- PHP: Shared Nothing
- Kein Zustand, außer in Caches
- Aufteilung der Funktionalität in Legacy und Neu-System
- Komplette Separierung in Prozesse

“Proxy”

- Kommunikation über HTTP auf lokaler Schnittstelle
- Requests gehen grundsätzlich an das neue Backend
- Proxy-Modul entscheidet über Forwarding
- Requests **und** Events werden über HTTP ausgetauscht
- Event-Modell erlaubt feingranulare Aufteilung

Proxy Setup

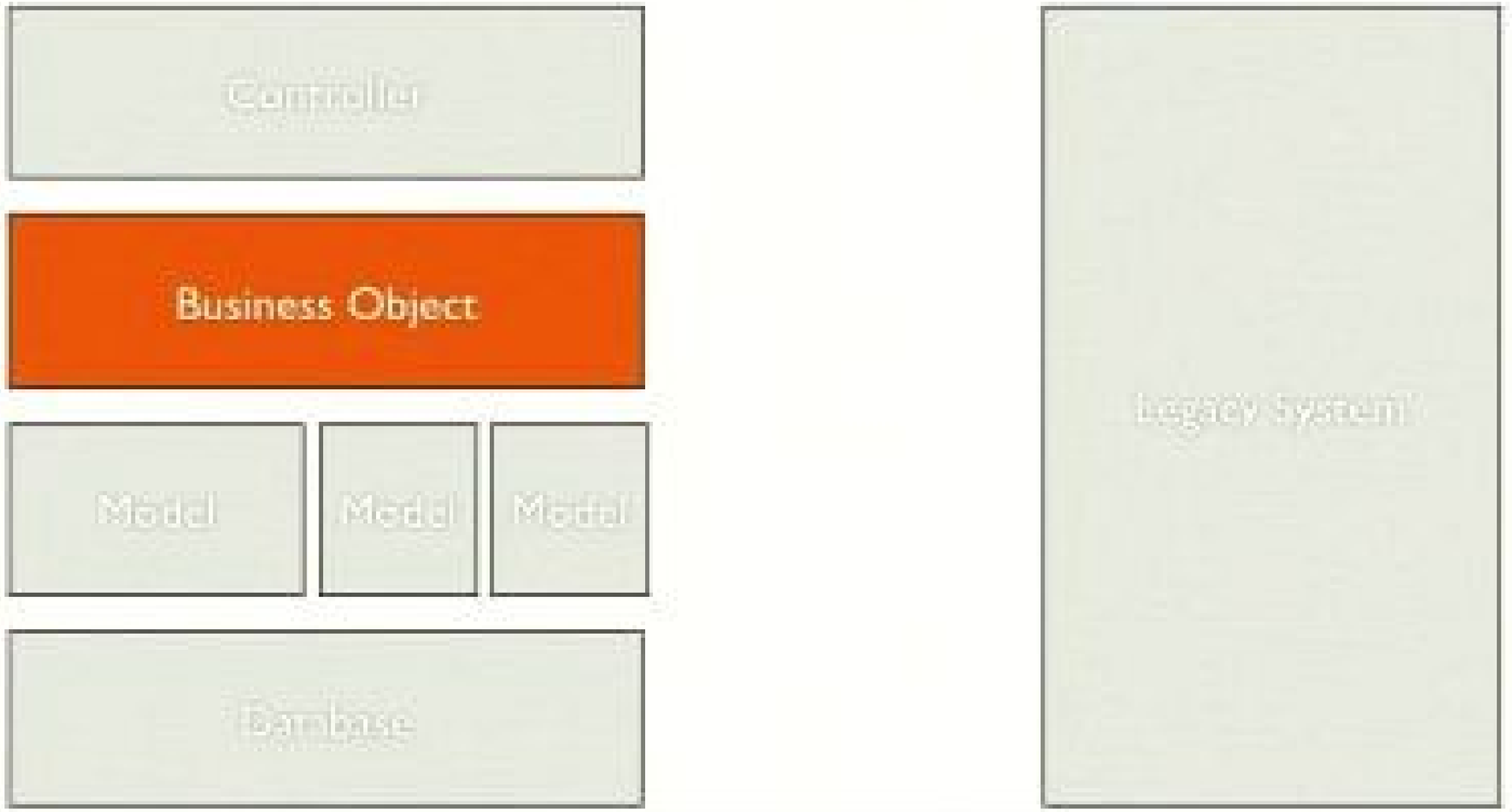
Proxy Setup


Frontend
Request-Proxy
Legacy Backend
Symfony2 Backend
Event-Proxy
Database

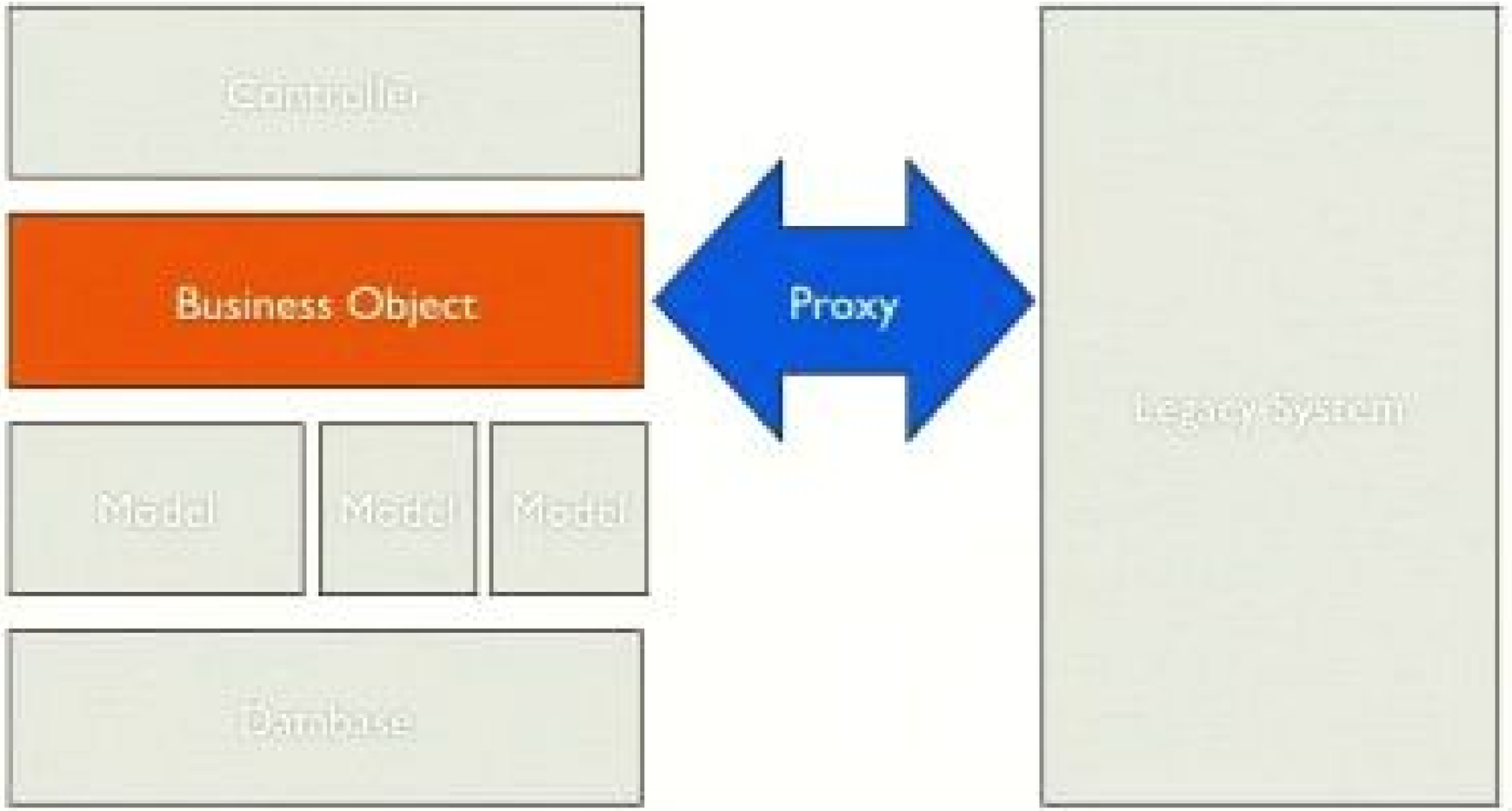
Kompatibilitätsschicht

- Interaktion erfordert Kompatibilität auf Model-Ebene
- Model-Schicht in Legacy-System unsauber
- Aggregation von Entities in serialisierbare "Business Objects"
- Trade-Offs können iterativ nach dem Rewrite aufgelöst werden

Kompatibilitätsschicht



Kompatibilitätsschicht



Probleme?

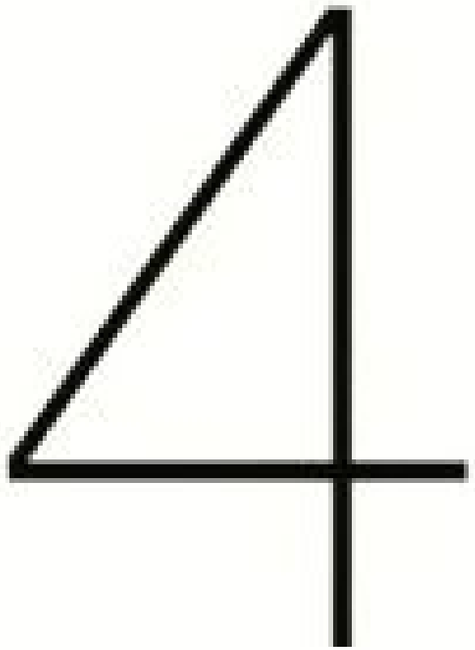
- Kommunikation funktioniert
- Strikte Trennung der Systeme verhindert Seiteneffekte
- Authentifizierung wird in beiden Systemen durchgeführt
- Geringe Performanceeinbußen für alte Funktionalität

Das Ergebnis

Rewrite > Refactoring

MANOffice® 4.0

- Voll getestetes System
- Moderne Web Plattform
- Klare Architektur
- Aufwand für Rewrite erheblich höher als geschätzt (aber auch für Refactoring)
- Rollout-Termin konnte trotzdem eingehalten werden



Q & A