Killing me softly

Refaktorieren von Monolithen

Hristiyan Pehlivanov
Mathema Software GmbH

Agenda

- Schlechte Ideen für Refactoring
- Eine bessere Idee für Refactoring
  - CI/CD
  - Extrahieren von Services
  - Verteilte Systeme
- Beispiel für ein großes Refactoring

MATHEMA

Impressum Kontakt

Suche

Legacy System

- Erfüllt nichr mehr die Geschäftsanforderungen
- Muss weiterentwickelt werden
- Wartung und Änderungen sind extrem aufwändig

Microservices

" In short, the microservice ***architectural style*** *[1] is an approach to developing a single application as a suite of* ***small services****, each running in its* ***own process*** *and* ***communicating with lightweight mechanisms****, often an HTTP resource API. These services are* ***built around business capabilities and independently deployable*** *by fully automated deployment machinery. There is a* ***bare minimum of centralized management*** *of these services, which may be written in different programming languages and use different data storage technologies."*

Martin Fowler

how to lose weight

how to lose weight **fast**
how to lose weight **in 1 week**
how to lose weight **by walking**
how to lose weight **in ramadan**
how to lose weight **in a month**
how to lose weight **in 2 weeks**
how to lose weight **without sport**
how to lose weight **in 3 months**
how to lose weight **fast for men**
how to lose weight **on face**

Google Search | I'm Feeling Lucky

Learn more

Report inappropriate predictions

refactor monolith

refactor monolith**ic to microservices**
refactor monolith
refactor**ing** monolith**ic to microservices with functional programming**
bluemix refactor monolith**ic applications**
how to refactor monolith**ic code**

Google Search | I'm Feeling Lucky

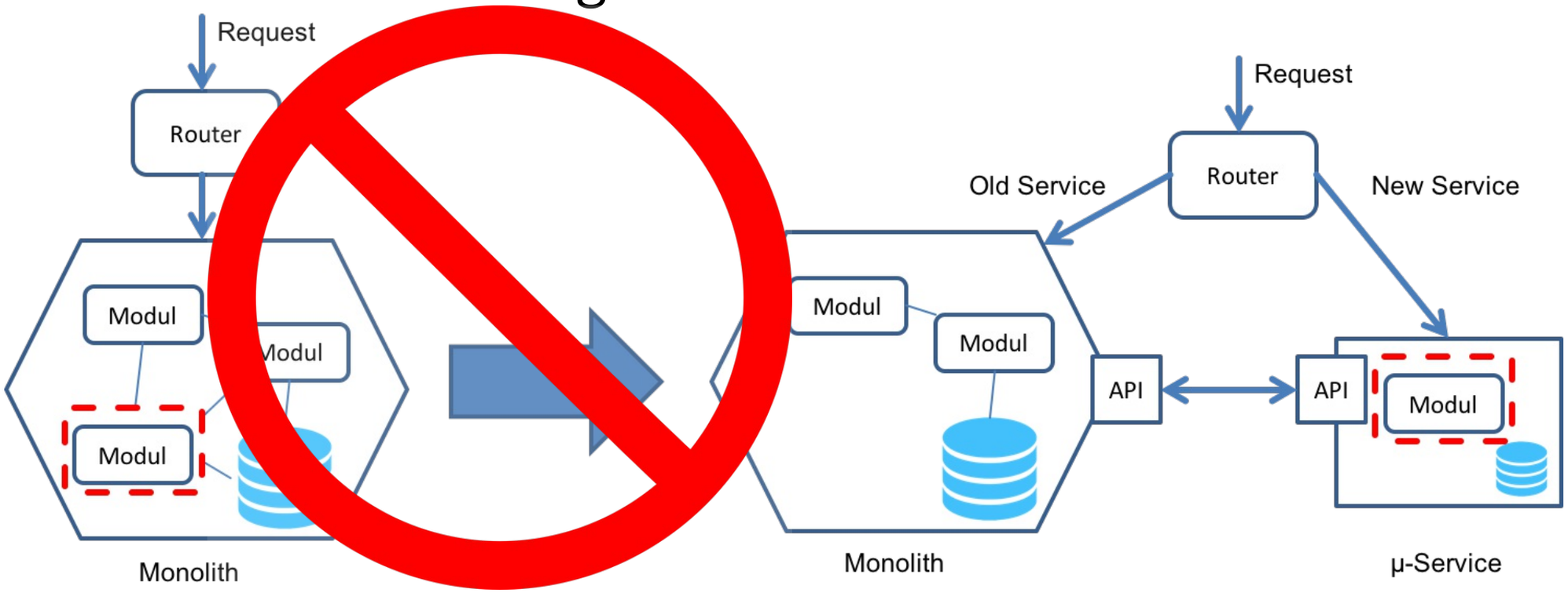
Learn more

Report inappropriate predictions

Schlechte Diäten

- Neuer Service erzeugen, Daten des Monolithen verwenden
- Frontend und Backend in eigenen Services aufteilen
- Zuerst ein Modul isolieren, danach das Modul als neuen Service extrahieren

Modul ≠ Selbstständiger Service



“Die Bezeichnung Diät kommt von altgriechisch δίαιτα díaita und wurde ursprünglich im Sinne von “Lebensführung”/“Lebensweise” verwendet.”

Wikipedia

Monolith refaktorieren

1. Schnelle Feedback-Loops vorbereiten (CI/CD)
2. Kerngeschäft extrahieren
3. Das verteilte System aufbauen

Endziel

- Selbstständige Services
- Unabhängig redeploybar

Notwendiges Vorwissen

- Doman Driven Design
- Bounded Context
- Self-Contained Systems
- Strangler Pattern

Phase 1: Training

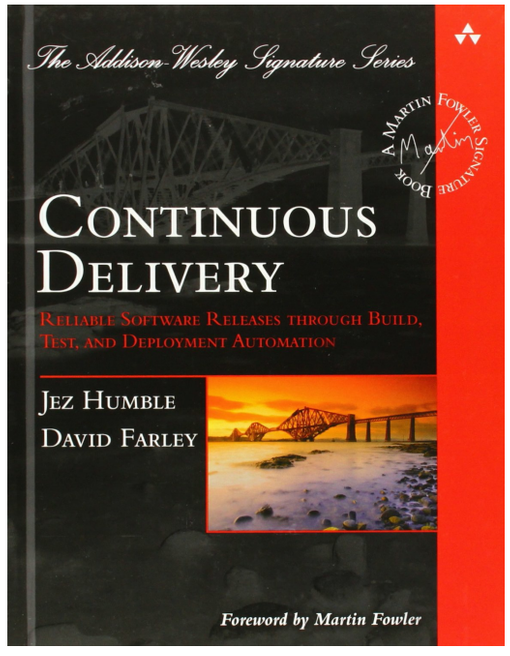
- Mit einem „einfacheren“ Service anfangen
  - Nicht kritisch fürs Geschäft
  - Kann vom Monolith leichter entkoppelt werden
  - Am besten ohne persistierte Daten
  - CI/CD-Vorlage für neue Services vorbereiten
- Fokus auf CI/CD, nicht auf Microservices

Phase 1: Continuous Integration

- Checkt jeder Entwickler täglich auf dem Main-Trunk ein?
- Können Sie sich auf Ihre automatisierten Tests verlassen?
- Ist ein roter Build die höchste Priorität des Teams?

Phase 1: Continuous Delivery

- Ist Deployment und Rollback automatisiert?
- Falls ich jetzt eine IF-Abfrage ändere, wie lange dauert es bis zum Deployment auf dem Produktionssystem?
- Ist jeder Commit releasefähig?



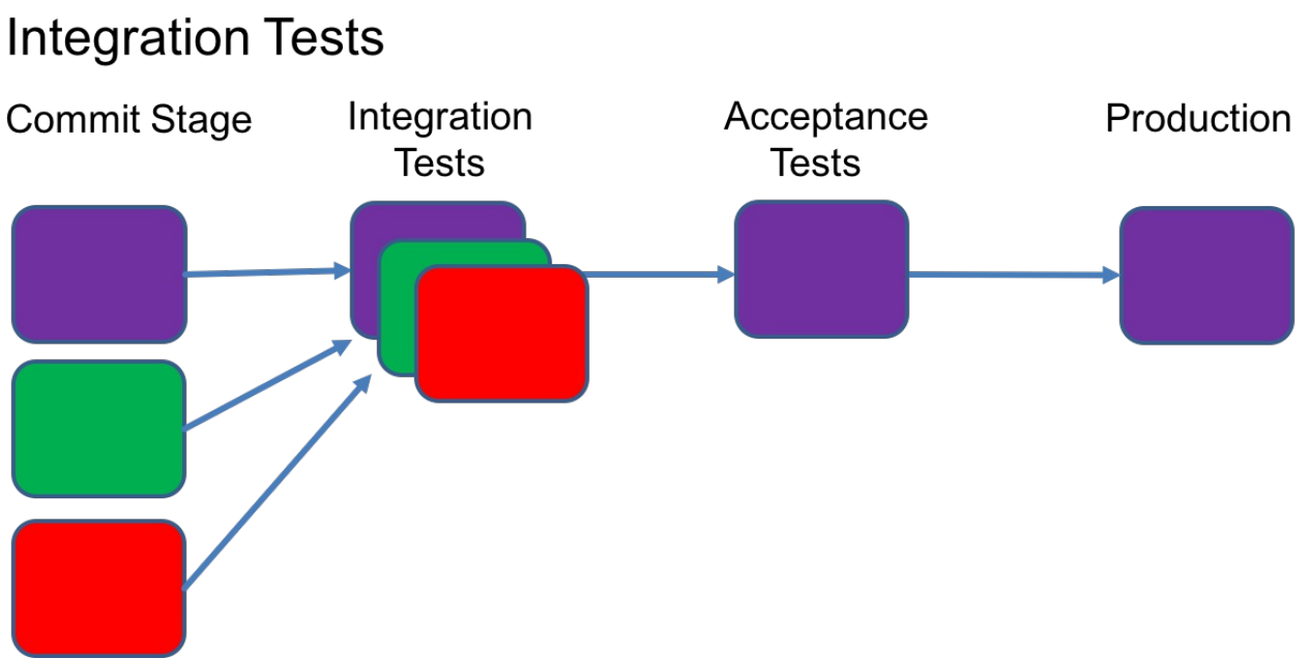
Testen von verteilten Systemen

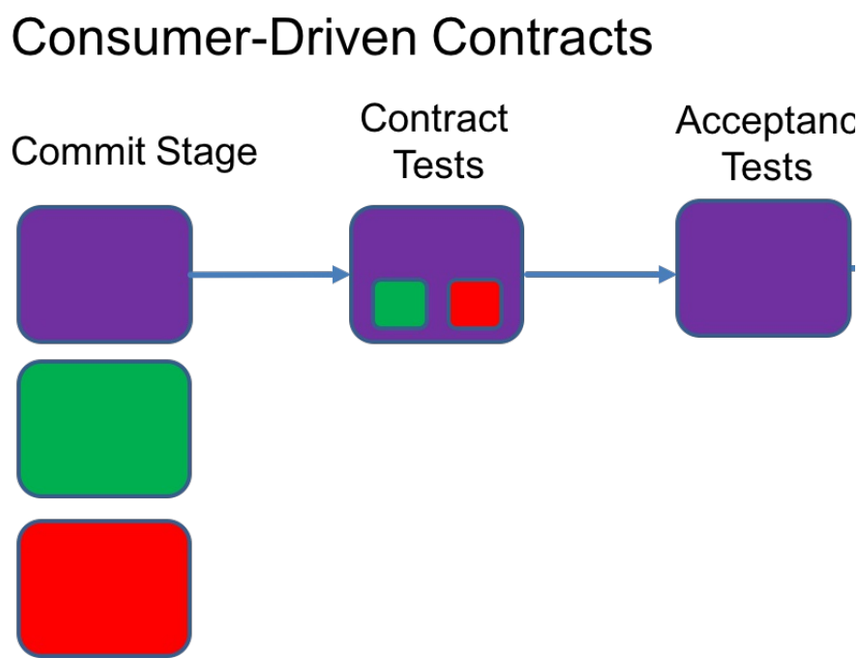
Mocks	Integrationstests	Consumer-Driven Contracts
Billig	Teuer	Billig
Schnell	Langsam	Schnell
Stabil	Fragil	Stabil
Nicht glaubwürdig	Glaubwürdig	Glaubwürdig

Consumer-Driven Contracts

- Consumer und Producer einigen sich auf einen Contract
- Contract wird von Consumer implementiert

  und bei Producer ausgeführt





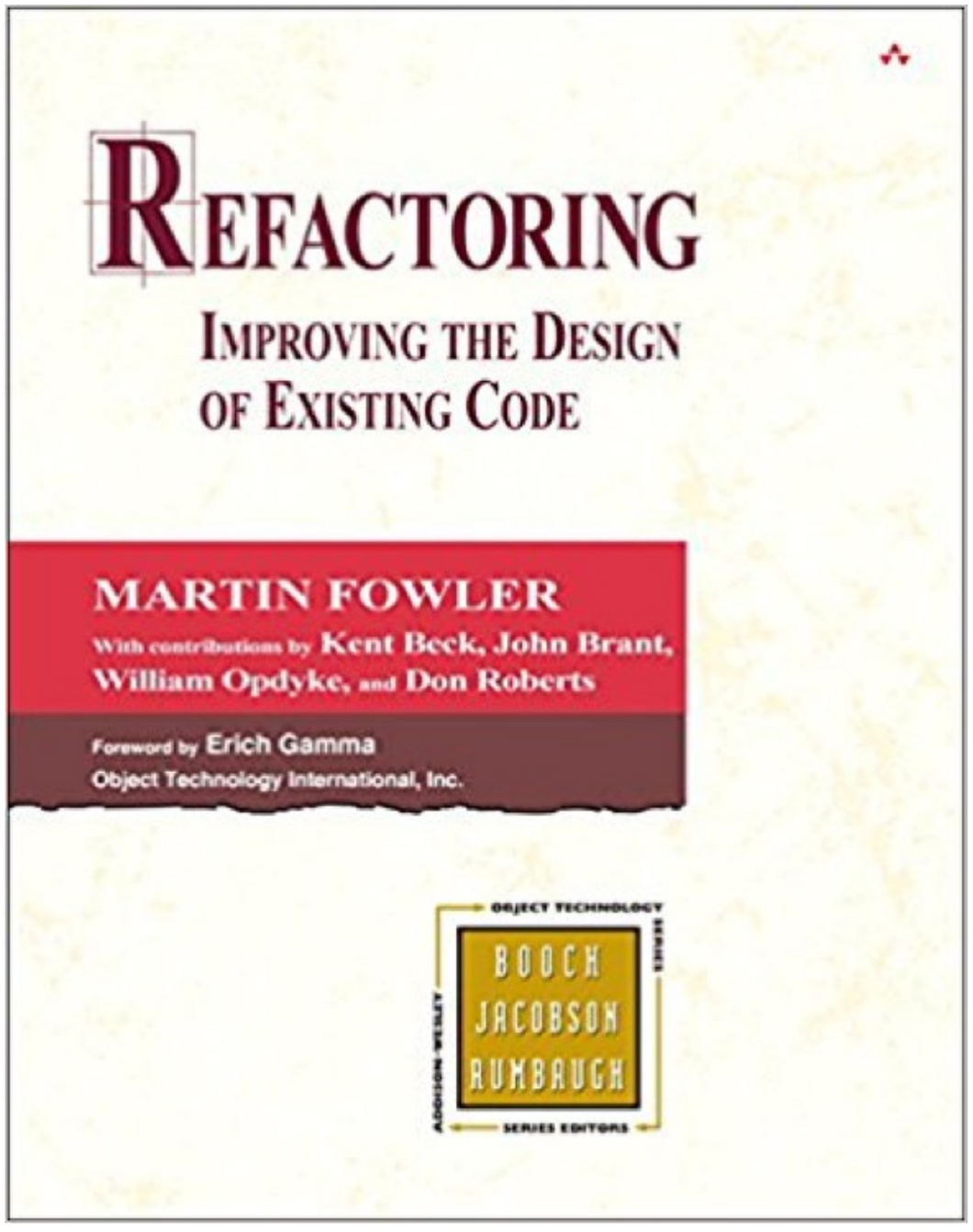
Phase 2: Kerngeschäft

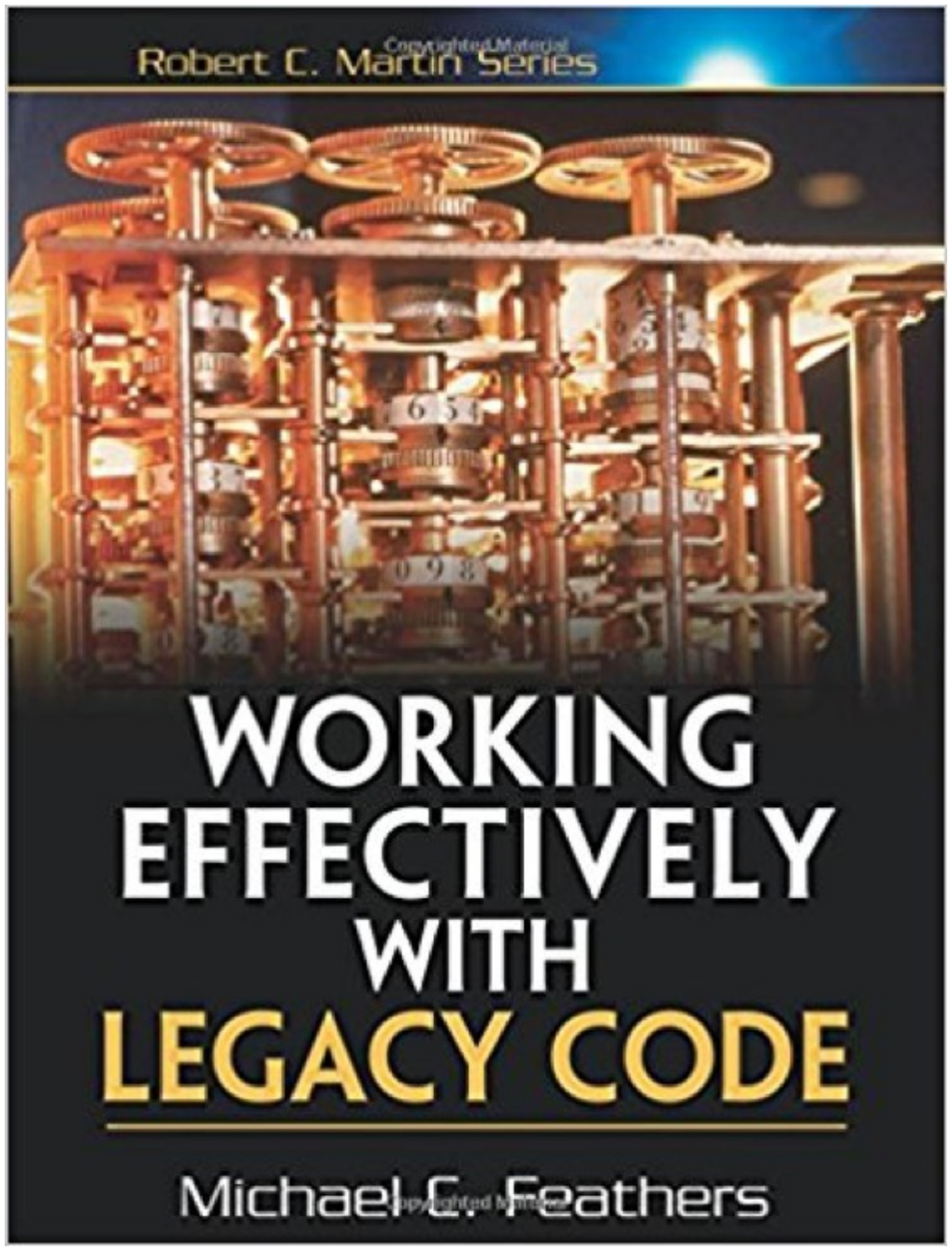
- Was ist der wichtigste Teil unserer Software

  den wir weiterentwickeln wollen?
- Extrem kritische Aufgabe, Risiko über CI/CD minimieren
- Vertikal arbeiten – einen Bereich von den Daten bis zum Frontend extrahieren

- If you are going to fail, fail fast!
- Fokus auf selbstständige Services, immer noch nicht auf Microservice

Refactoring



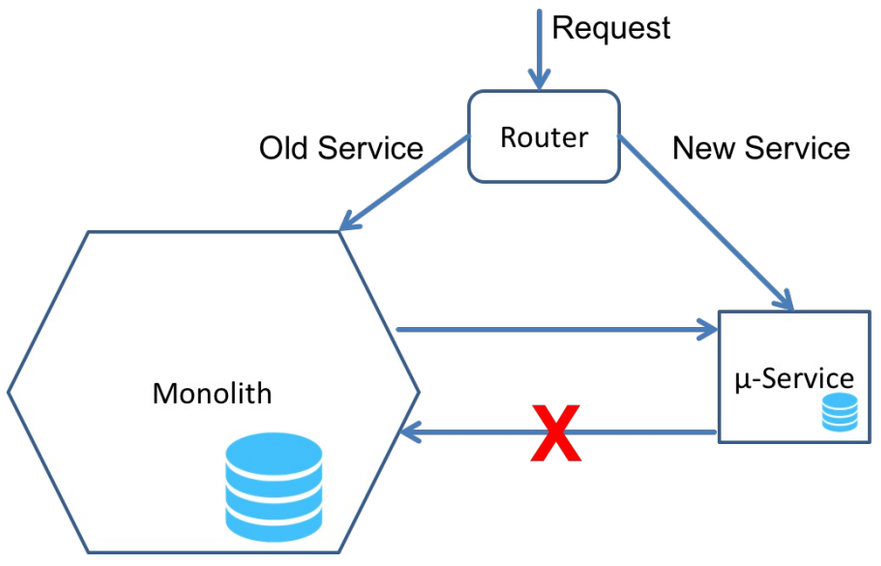


Refactoring

- Tests vorbereiten
- Kleine Änderung vornehmen und über Tests verifizieren
  - Falls Tests grün - OK
  - Falls Tests rot - Revert & Repeat
- Entweder Refaktorieren ODER neues Feature implementieren
- Code nicht einfach wegwerfen, schlechter Code ≠ wertloser Code

Abhängigkeiten

- Der Monolith ruft unseren neuen Service auf
  Zuerst solche Services extrahieren
- Umgekehrt machen wir uns vom Monolith abhängig
- Falls doch nicht möglich, Anti Corruption Layer verwenden
- Versteckte Abhängigkeiten im globalen Kontext (Session, usw.) beachten

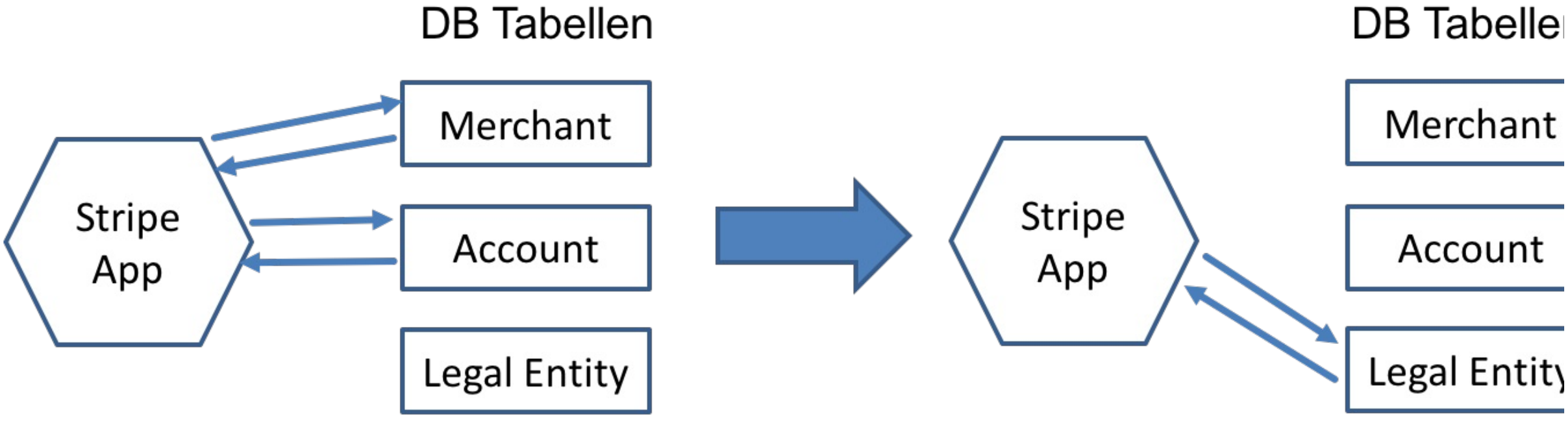


Datenbank

- Ein Service ist nur dann selbstständig, wenn er seine Daten beherrsch
- Services stellen Funktionen zur Verfügung, keine Daten
- OO-Prinzipien auf Systemebene

Stripe

- Online-Bezahldienst
- Wegen Geschäftsausweitung musste das DB-Schema geändert werde
- Produktionssystem darf nicht gestört werden (Bezahldienst)
- Eventuelle Fehler könnten auch rechtliche Konsequenzen haben



https://robertheaton.com/2015/08/31/migrating-bajillions-of-databas

records-at-stripe/

Stripe 4-Phasen Datenmigration

1. **Datenmigration**
   - Double-Writing implementieren
   - Daten migrieren
2. **Lese-Zugriffe über Proxy umleiten**
   - Von der neuen Tabelle lesen
   - Schreiben in den alten Tabellen einstellen
3. **Lese- und Schreib-Zugriffe direkt in der neuen Tabelle**
   - Alte Lese- und Schreib-Zugriffe aufräumen
   - Über Logging Nachzügler finden
   - Proxy entfernen
4. **Cleanup**
   - Über Logging Aufrufe zu den alten Tabellen finden (Speichern, usw
   - Produktionslogs überprüfen
   - Finales Cleanup

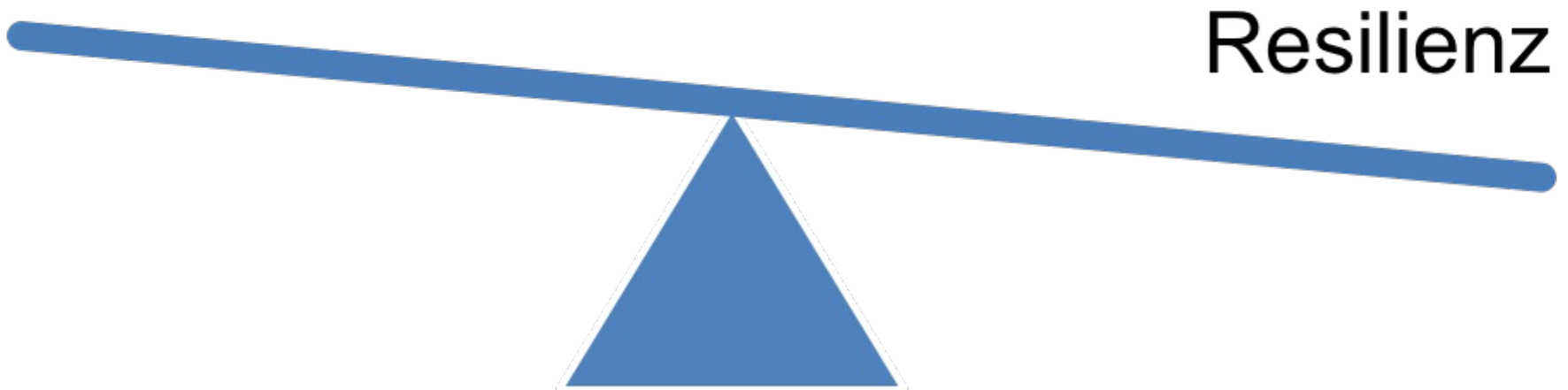
Phase 3: Verteiltes System

- Jetzt stehen selbstständige Services zur Verfügung
- Feedback zum Systemzustand über CI/CD gesichert
- Falls notwendig kann man jetzt ein verteiltes System aufbauen
  - Monitoring
  - Load Balancing
  - Circuit Breakers
  - usw.

Micro vs Macro Architektur

Skalierbarkeit
Autonomie
Flexibilität

Kommunikation
Betrieb
Resilienz



Fazit

- Ein großes Refactoring besteht aus vielen kleinen Schritten
- Funktionen schrittweise extrahieren und neben dem Monolith betreiben
- CI/CD ist die notwendige Grundlage für komplexe Systeme
- Nur Code in Produktion ist “done”
- Das verteilte System ist der letzte Schritt, nicht der erste

Danke für die Aufmerksamkeit!

Hristiyan Pehlivanov

Mathema Software GmbH

hristiyan.pehlivanov@mathema.de