













































# Tipps & Tricks für das Testen von Microservices

Jörg Pfründer

Hypoport AG

# EUROPACE

ANGEBOTE		Angebote aushändigen		Angebote aktualisieren				
   	Soll / Effektiv	Beleihungs- auslauf	Produktanbieter	Monatliche Gesamtrate	Darlehens- summe	Zins- bindung	Angebotsfrist	
   								 
  	2,04 % / 2,06 %	55,56 %	 VR-Musterbank	337 €	100.000 €	10 J.	07.04.2014	
  	2,35 % / 2,38 %	55,56 %	 VR-Musterbank inkl. 50.000 € KfW 124	 356 €	100.000 €	10 J.	07.04.2014	
  	2,12 % / 2,14 %	66,67 %	 VR-Musterbank	412 €	120.000 €	10 J.	07.04.2014	
  	3,03 % / 3,08 %	55,56 %	 Musterbank inkl. 50.000 € KfW 124	 412 €	100.000 €	10 J.	07.04.2014	
  	3,40 % / 3,45 %	55,56 %	 Musterbank	450 €	100.000 €	10 J.	07.04.2014	
  	3,81 % / 3,88 %	55,56 %	 Musterbank	484 €	100.000 €	15 J.	07.04.2014	



# EUROPACE

15% der  
Immobilienkredite  
Deutschlands



# EUROPACE

15% der  
Immobilienkredite  
Deutschlands

ca. 3 Mrd Euro / Monat

# Technologien

Java / Groovy

MongoDB (Oracle)

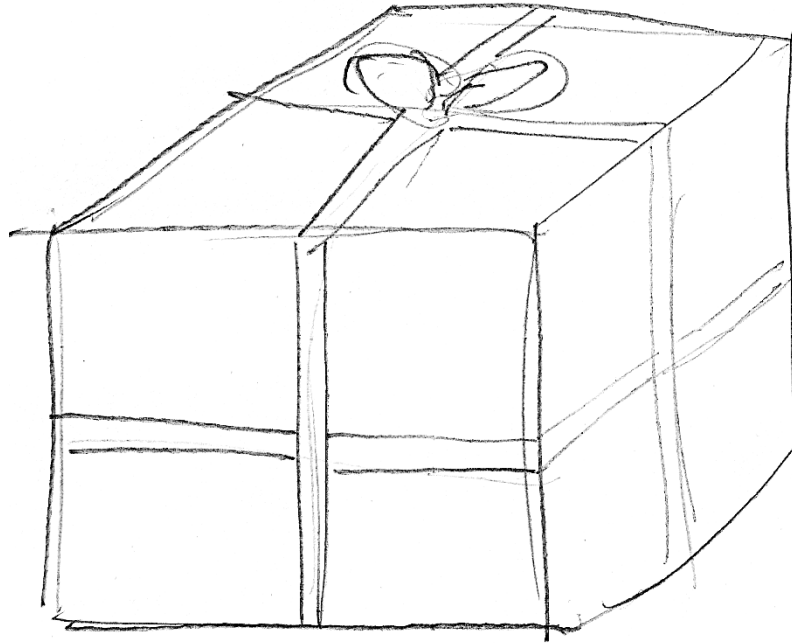
Spring

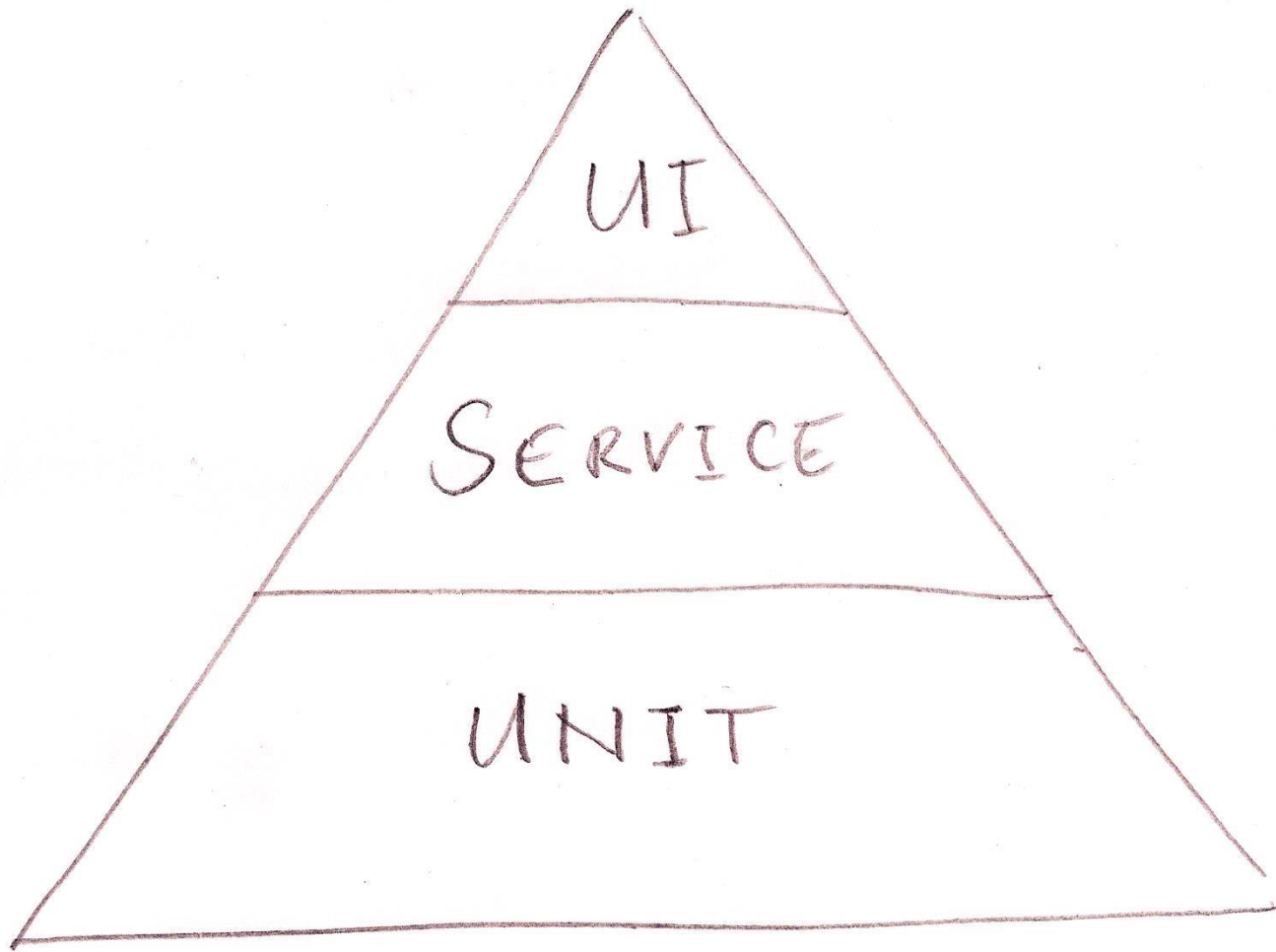
Tomcat / Spring Boot

# Agenda

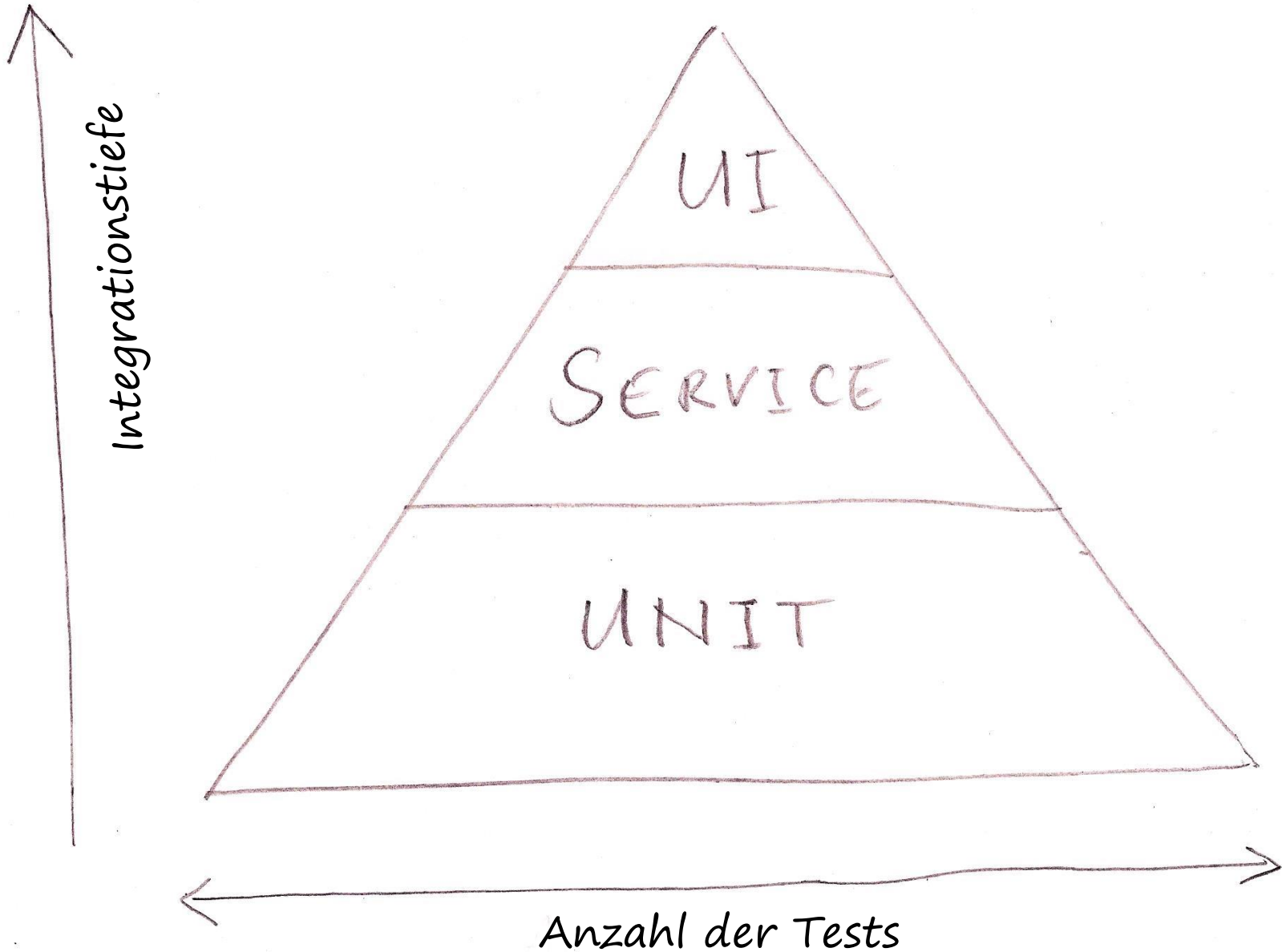
- Traditionelle Testpyramide
- Besondere Herausforderungen bei Microservices
- Postel's Law für Tests
- Drück' Dich aus!
- Production Code ist Test Code
- Täusche und betrüge
- Doppelt hält besser

# Monolith

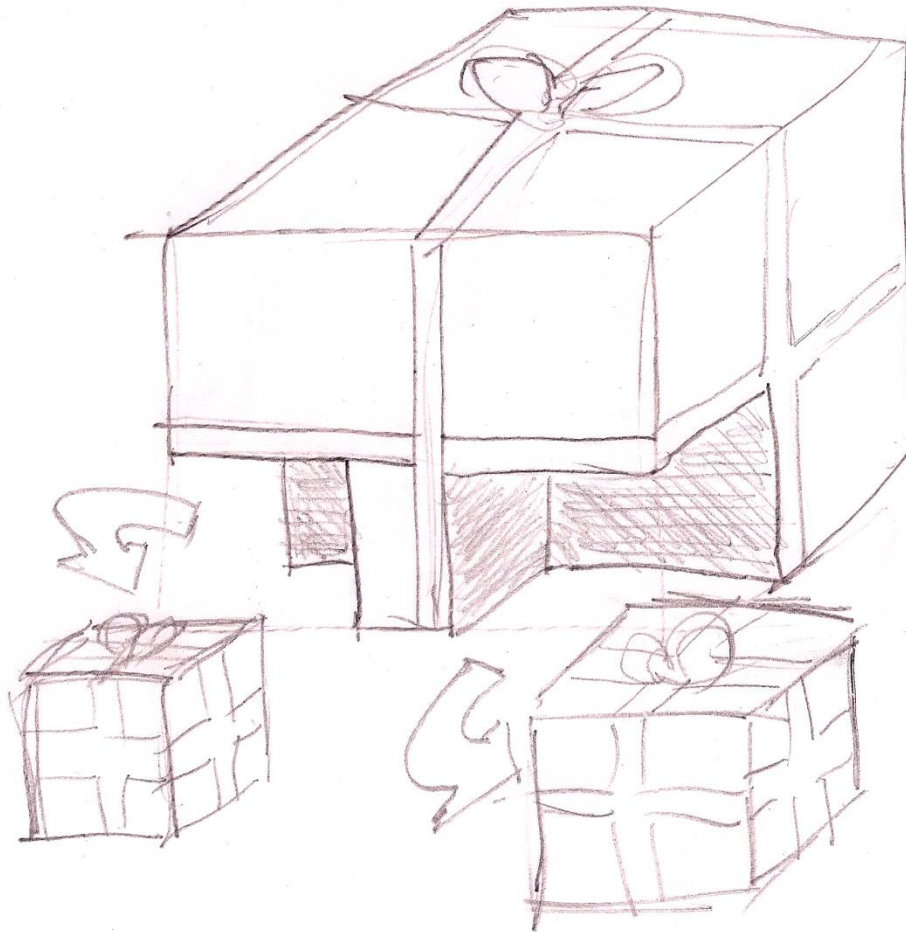


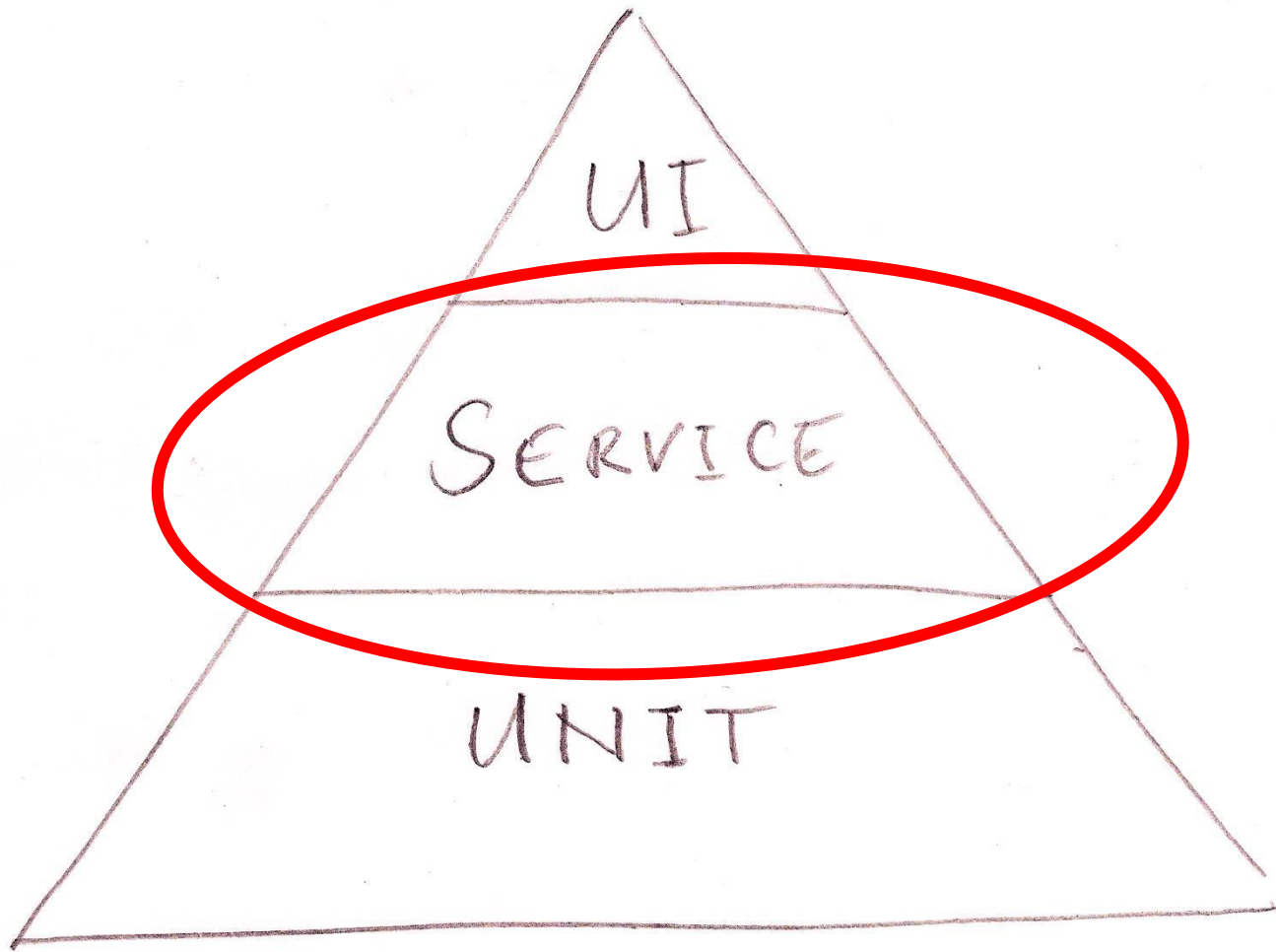






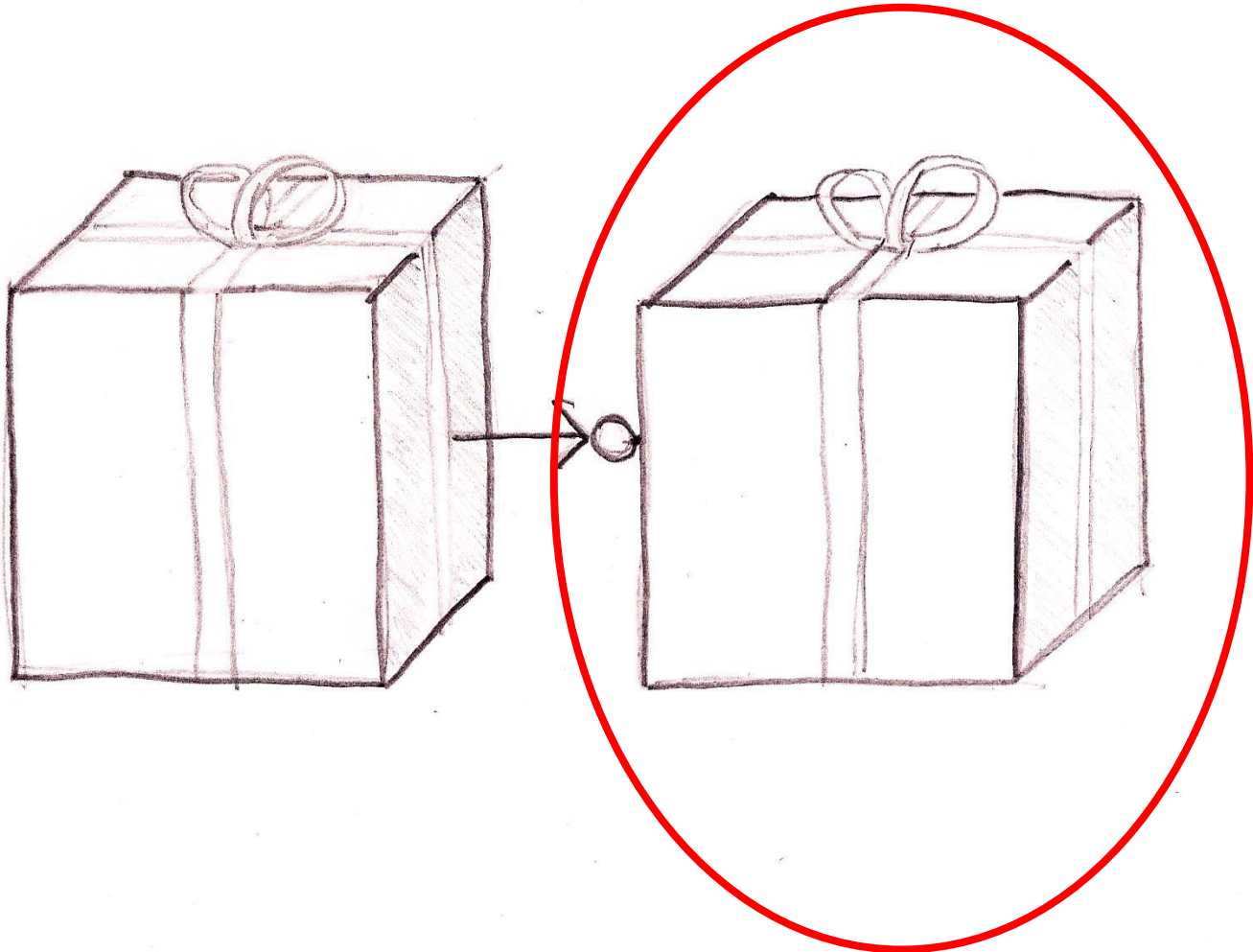
# Lose Kopplung





#1

# Postel's Law für Tests



# Problem #1

```
{„vorname“ : „Max“, „nachname“ : „Mustermann“ }
```

*String equals*

# Problem #1

```
{  
  „vorname“   : „Max“,  
  „nachname“  : „Mustermann“  
}
```

~~String equals~~

JSONassert (XMLAssert)

# Problem #1

```
{  
  „vorname“   : „Max“,  
  „nachname“  : „Mustermann“,  
  „name“      : „Max Mustermann“  
}
```

~~String equals~~

JSONassert (XMLAssert) ??

# Problem #1

```
{  
  „id“ : 9586729475,  
  „vorname“ : „Max“,  
  „nachname“ : „Mustermann“,  
  „name“ : „Max Mustermann“  
}
```

~~String equals~~

~~JSONassert (XMLAssert)~~



# Inspiration: UI Automation

- Nur wenige Elemente der UI werden getestet
- Das meiste bleibt ungetestet

# Trick #1

```
{  
  „id“ : 9586729475,  
  „vorname“ : „Max“,  
  „nachname“ : „Mustermann“,  
  „name“ : „Max Mustermann“  
}
```

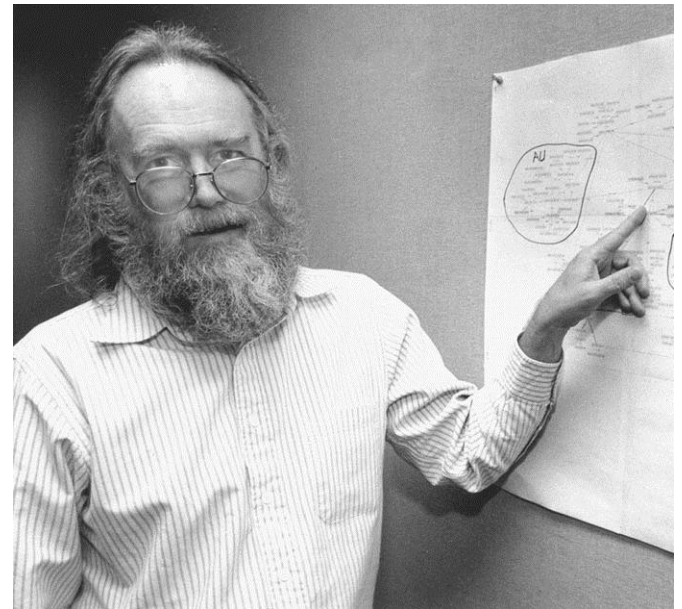
- *JSONpath (Xpath)*
- *Groovy RESTClient (SOAPClient)*

# Trick #1: Postel's Law

Be conservative in what you do,  
be liberal in what you accept  
from others.

Jonathan Postel

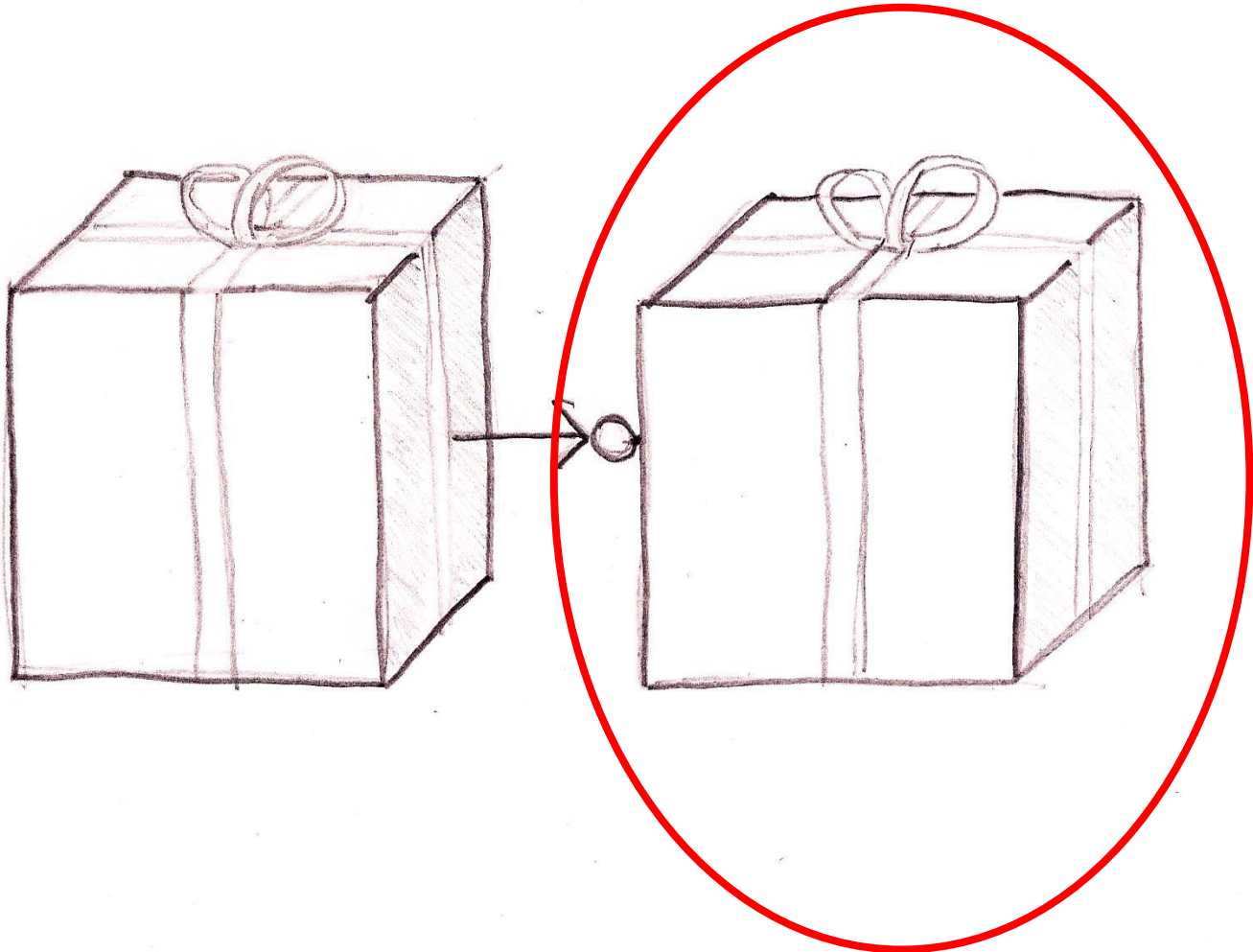
RFC793



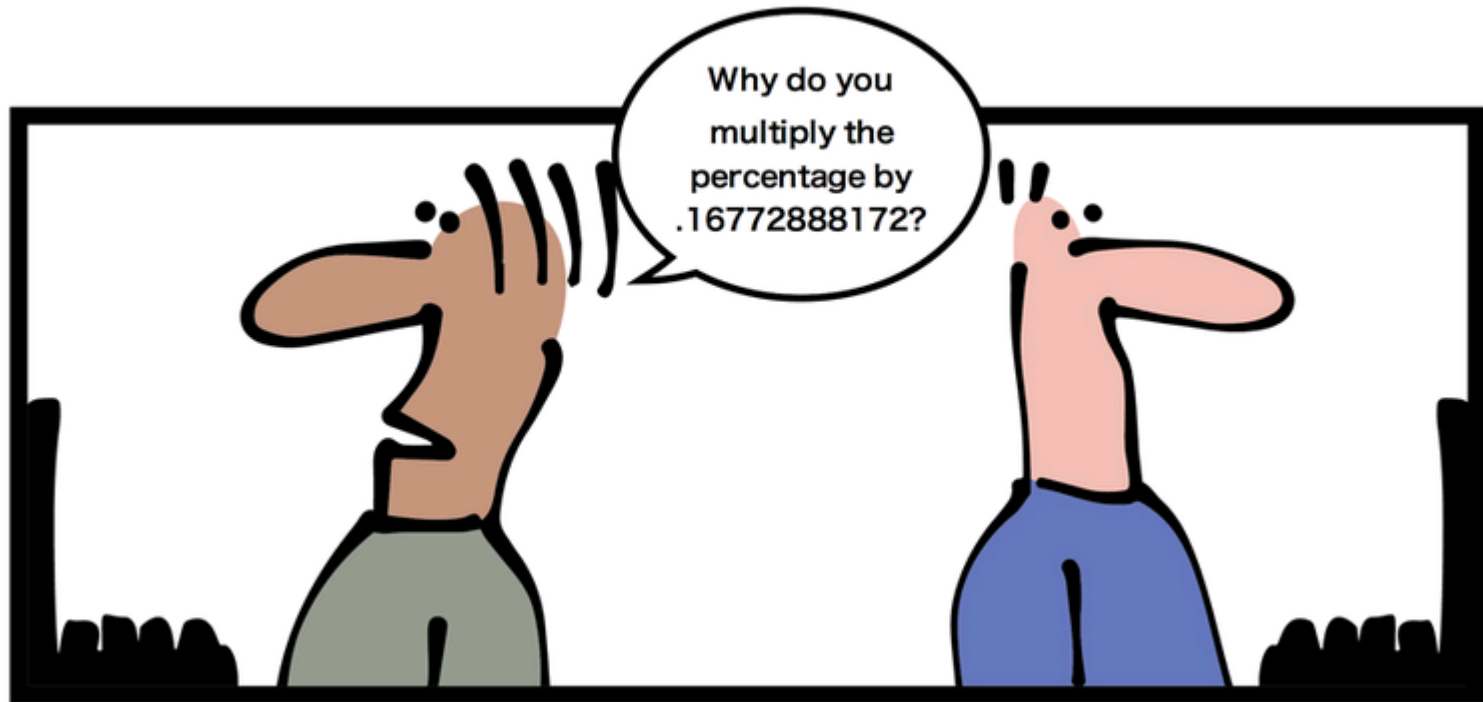
Lizenziert unter Attribution über Wikimedia Commons  
[http://commons.wikimedia.org/wiki/File:Jon\\_Postel.jpg#mediaviewer/File:Jon\\_Postel.jpg](http://commons.wikimedia.org/wiki/File:Jon_Postel.jpg#mediaviewer/File:Jon_Postel.jpg)

#2

Drück' Dich aus!

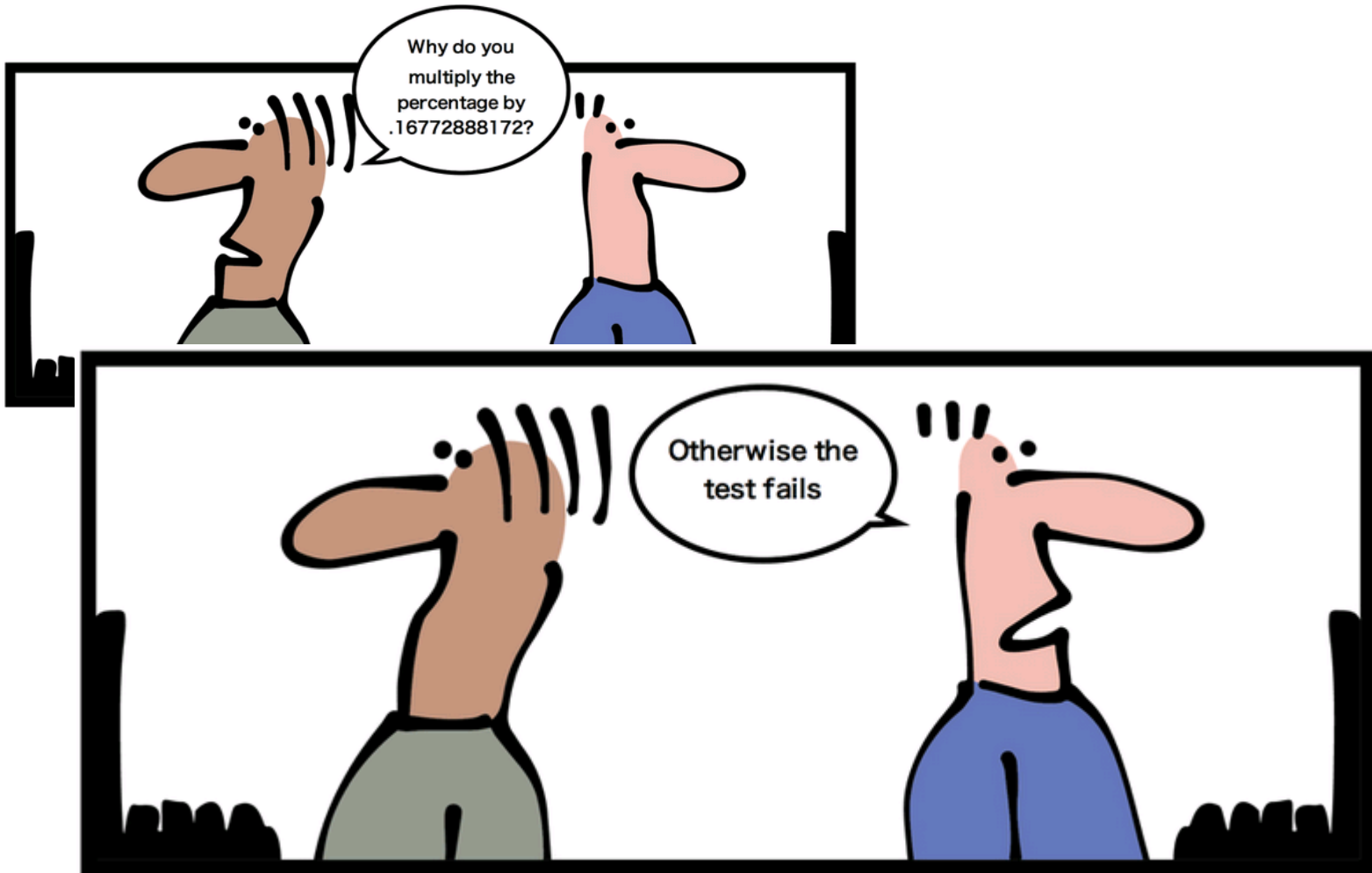


# Problem #2



<http://geek-and-poke.com/geekandpoke/2013/7/28/tdd>  
(Lizensiert unter Creative Commons 3.0 – mittleres Bild entfernt)

# Problem #2



# Inspiration: Webdriver Page Object

```
finanzielleSituation.gibRatenkreditEin(1,  
                                         RATE_75_EURO,  
                                         RESTSCHULD_5000_EURO,  
                                         "Hausbank");
```

# Trick #2: Response Object

- Antwort des Services in einem Response-Object kapseln
- Response-Object bietet fachliche Methoden an, macht intern die Umsetzung auf JSON bzw. XML



# Trick #2: Response Object

```
assert benutzerRequest.isSuccessful() == true
```

```
def response = benutzerRequest.response
```

```
assert response.vorname == "Max"
```

```
assert response.nachname == "Mustermann"
```

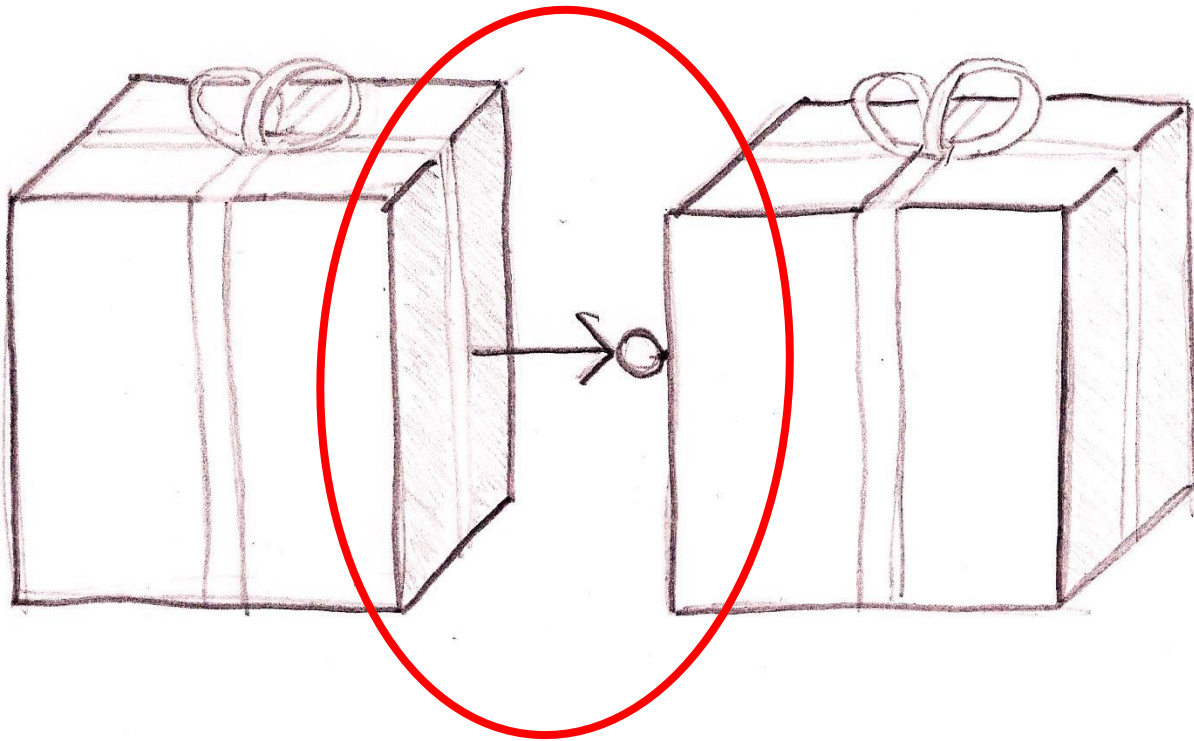
# Trick #2: Response Object

Verständlichkeit

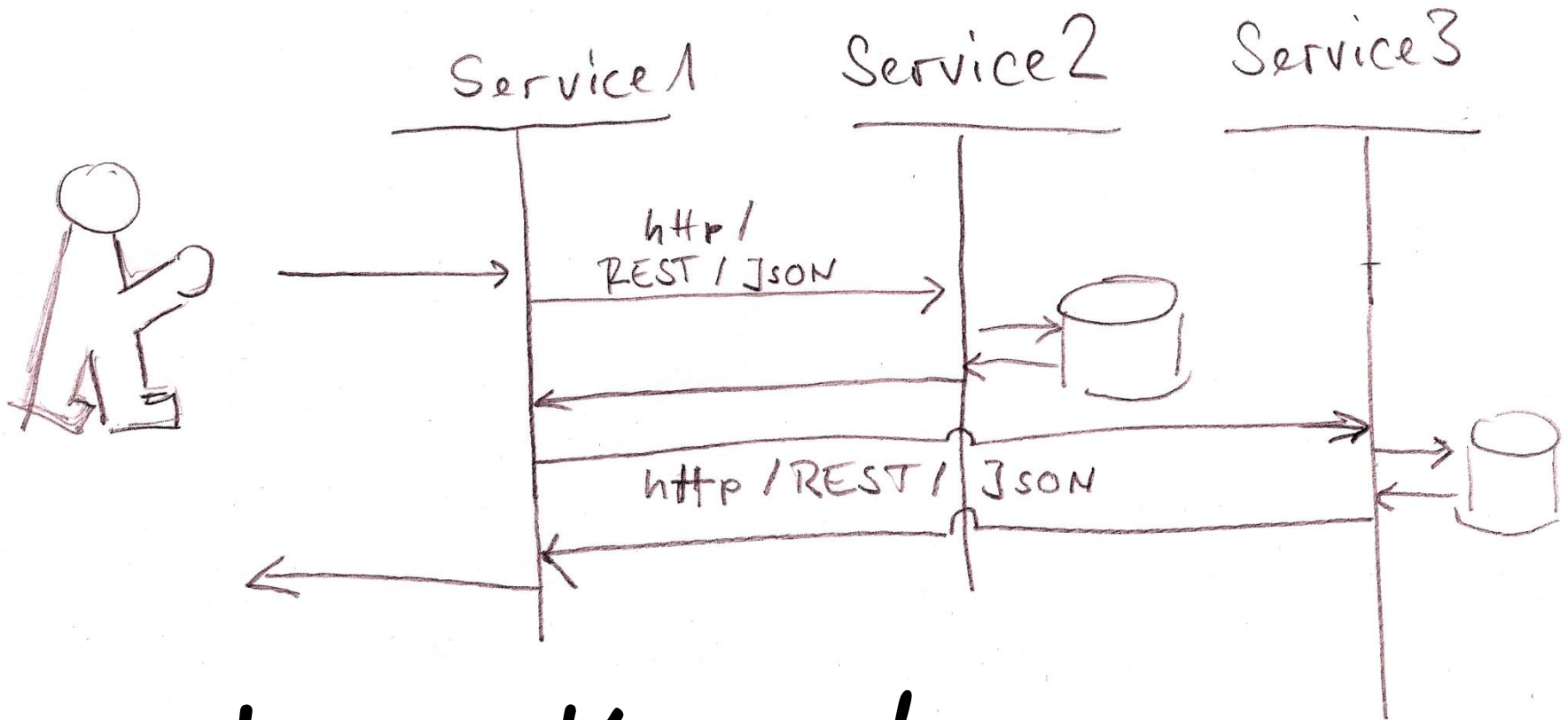
Wiederverwendung

#3

Production Code ist  
Test Code



# Problem #3



Lose Kopplung

## Trick #3

# Contract Tests

```
{  
  „id“ : 9586729475,  
  „vorname“ : „Max“,  
  „nachname“ : „Mustermann“,  
  „name“ : „Max Mustermann“  
}
```

Trick #3:

Production Code ist Test Code

Benutze die Implementierung des  
Clients als Test:

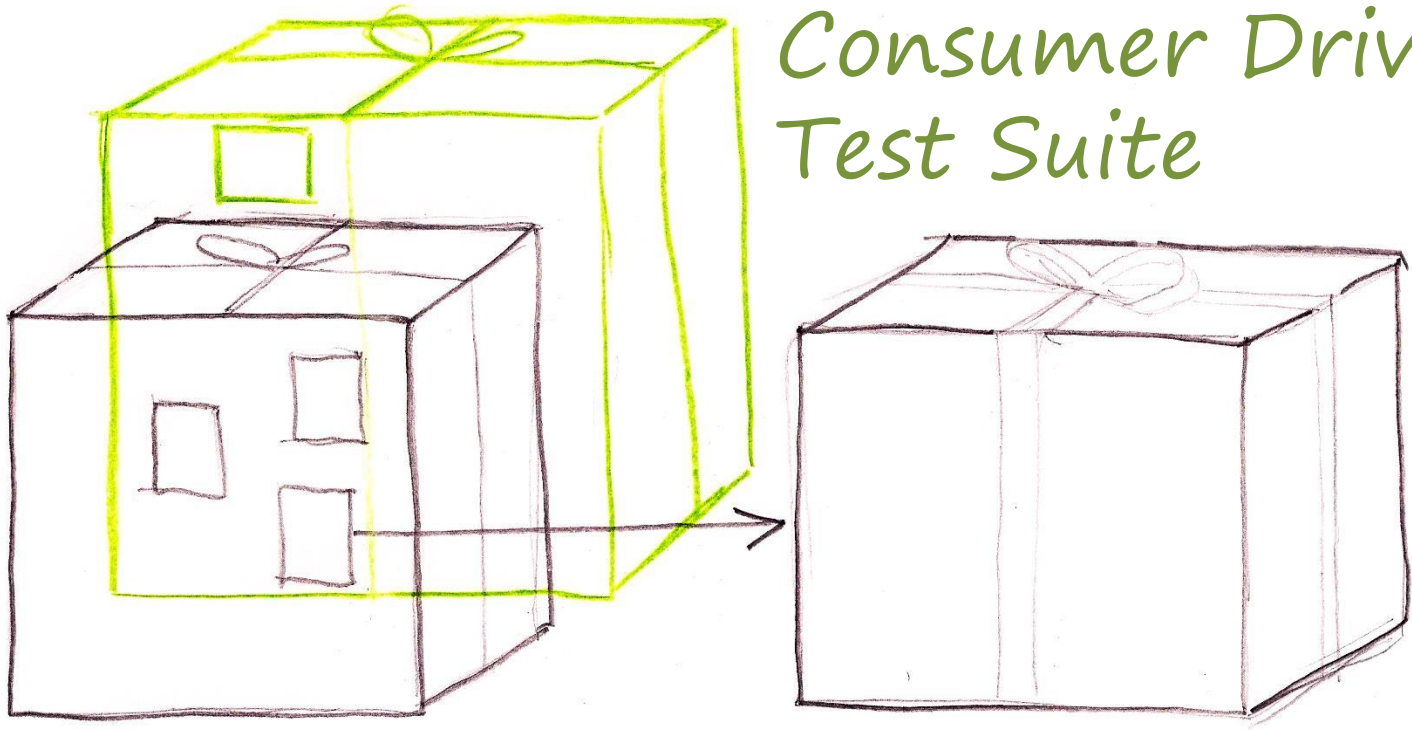
Consumer Driven Tests

bzw.

Consumer Driven Test Suite (CDTS)

# Trick #3: Production Code ist Test Code

Consumer Driven  
Test Suite



Consumer  
 $\mu$ -Service

Service-  
Provider

Trick #3:  
Production Code ist Test Code

Consumer Driven Tests

Technologieunabhängig?



# Trick #3:

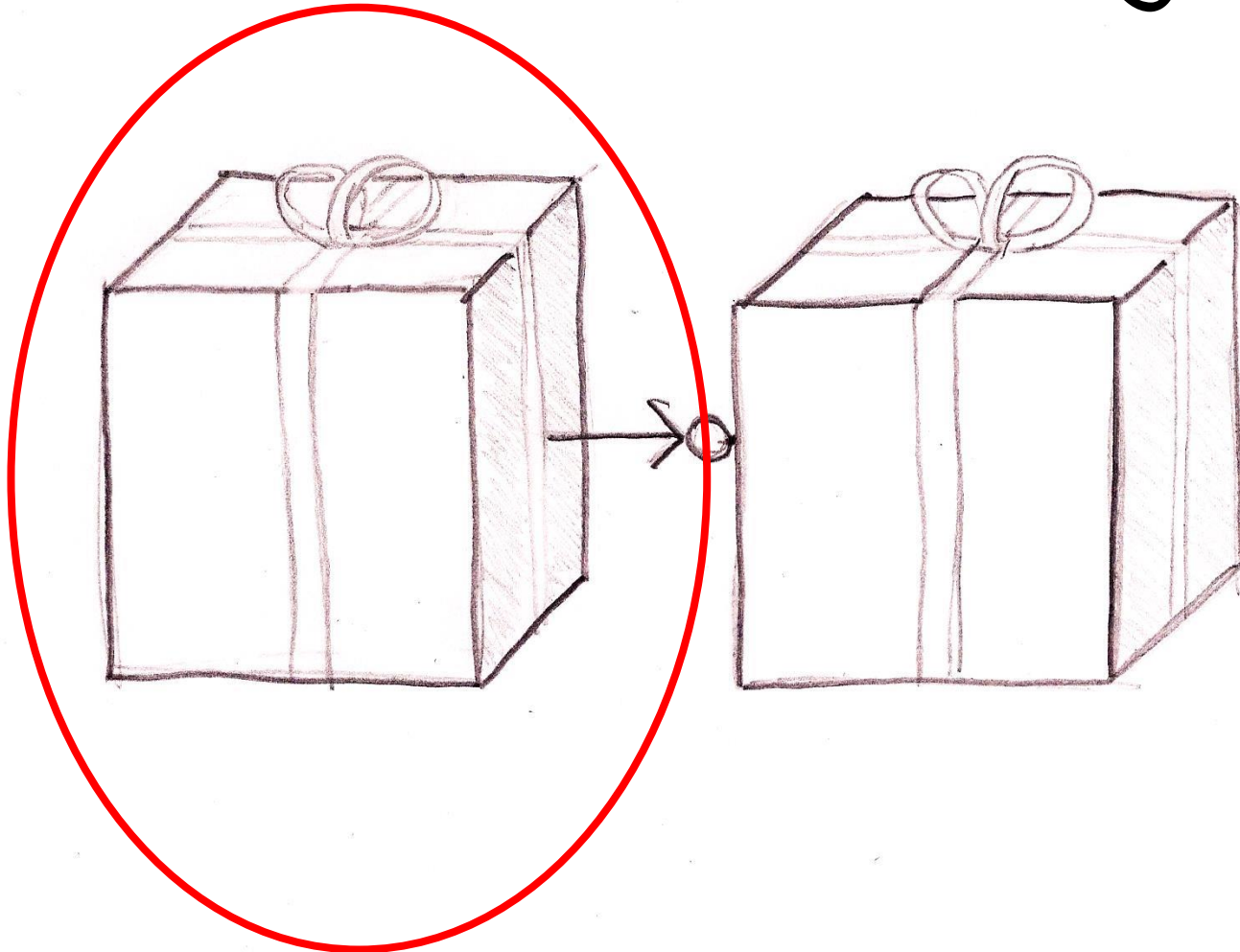
## Production Code ist Test Code

### beinahe technologieunabhängig:

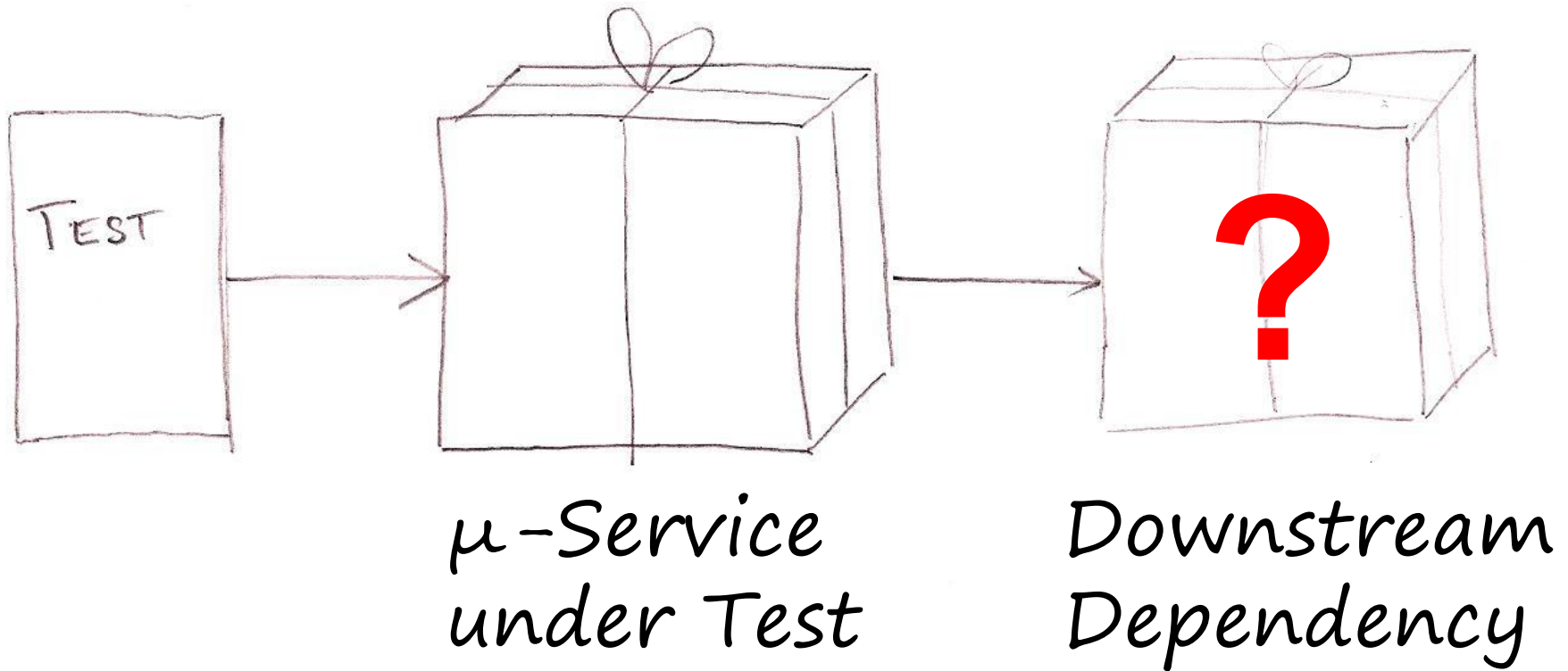
```
#!/bin/bash
waitUntilSUThochgefahren
curl -s "$TEST_ARTEFAKT_URL" > "cdts.jar"
java $PROPERTIES -jar cdts.jar
RESULT=$?
rm -f cdts.jar
if [ "$RESULT" -ne "0" ]; then
    error "Contract tests failed!!!"
fi
```

#4

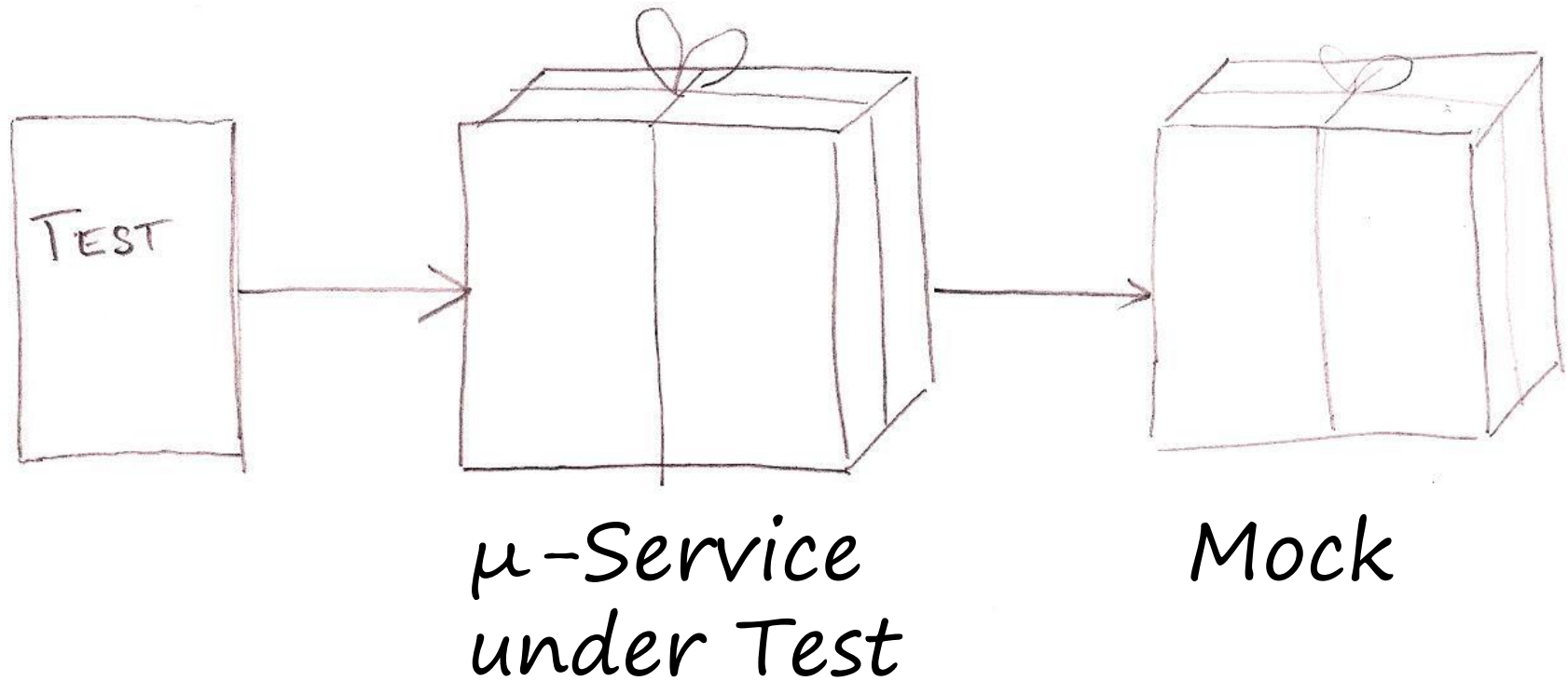
Täusche und betrüge!



# Problem #4: Downstream Dependencies



# Trick #4: Täusche und Betrüge

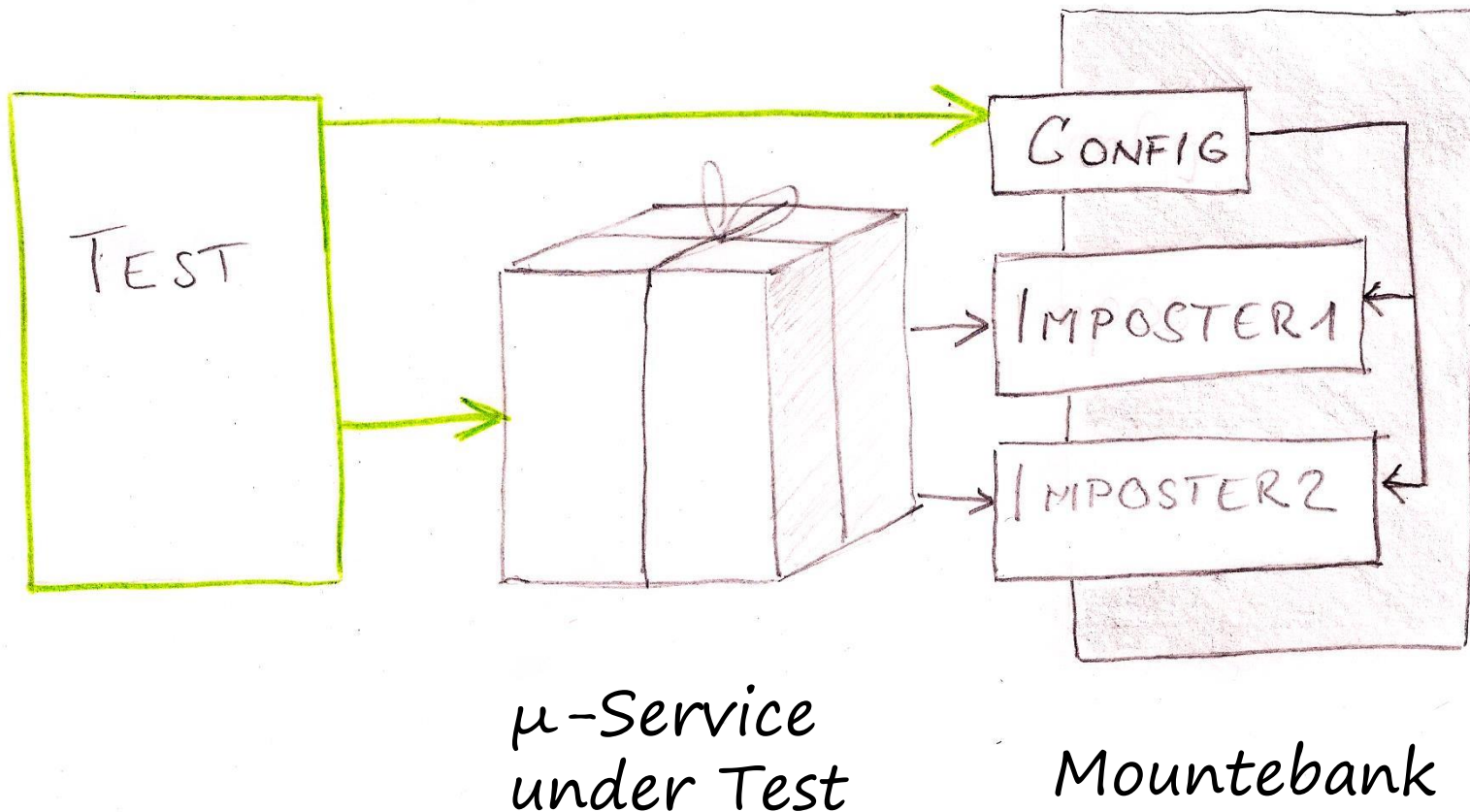


# #4: Mountebank – der Marktschreier

- [www.mbtest.org](http://www.mbtest.org)
- node.js basiert
- Erhältlich als standalone-  
Installationspaket oder via npm



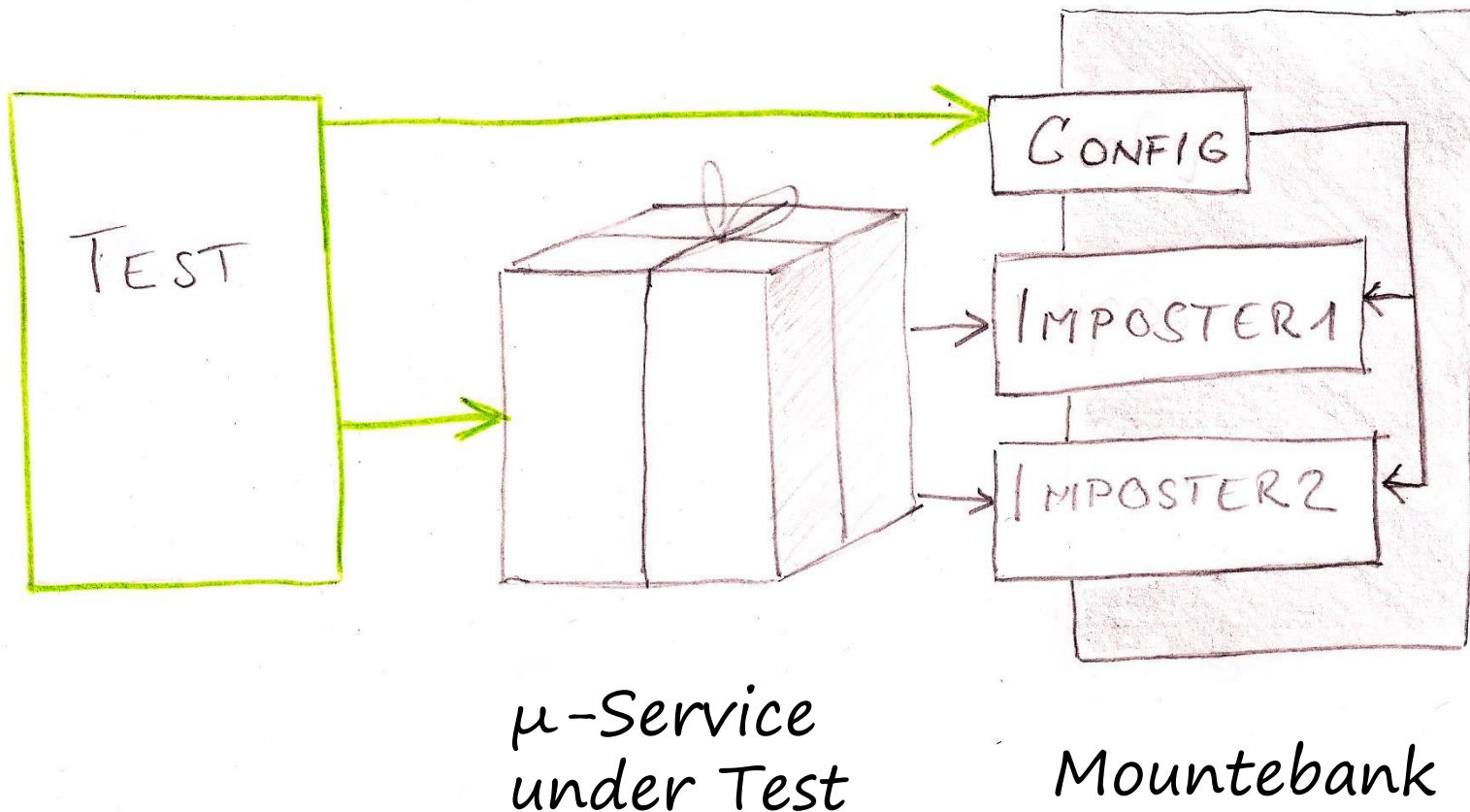
# #4: Mountebank - der Marktschreier



POST /imposters

```
{
  "port": 4545,
  "protocol": "http",
  "stubs": [
    {
      "responses": [
        {
          "is": {
            "statusCode": 200,
            "headers": {
              "Content-Type": "text/html; charset=utf-8"
            },
            "body": "<html><body><h1>Test</h1></body></html>"
          }
        }
      ],
      "predicates": [
        {
          "startsWith": {
            "path": "/dokumente/download"
          }
        }
      ]
    }
  ]
}
```

# #4: Mountebank - der Marktschreier





# #4: Täuscher auf Draht – Wiremock

- [www.wiremock.org](http://www.wiremock.org)
- Java-basiert
- als Bibliothek einbinden
- Konfiguration als nativer Java-Methodenaufruf
- kein Config-Port nötig

# #4: Täuscher auf Draht – Wiremock

```
WireMockConfiguration config = wireMockConfig().port(startPort);  
WireMockServer wireMockServer = new WireMockServer(config);  
wireMockServer.start();  
int port = wireMockServer.port();
```

```
stubFor(get(urlMatching("/dokumente/download/[A-Za-z0-9]+"))  
    .willReturn(aResponse()  
        .withStatus(200)  
        .withHeader("Content-Type", "text/html; charset=utf-8")  
        .withBody("<html><body><h1>Test</h1></body></html>"))));
```

# Wer startet Wiremock?

Der Test

Die Applikation

# Wer startet Wiremock?

## Der Test

- Mocks und Testfälle gehören logisch zusammen
- Separation von Testcode und Productioncode

## Die Applikation

# Wer startet Wiremock?

## Der Test

- Mocks und Testfälle gehören logisch zusammen
- Separation von Testcode und Productioncode
- Kompliziertes Setup

## Die Applikation

# Wer startet Wiremock?

## Der Test

- Mocks und Testfälle gehören logisch zusammen
- Separation von Testcode und Productioncode
- Kompliziertes Setup

## Die Applikation

- Weniger Wartezeiten beim Hochfahren
- Weniger Konfigurationsaufwand
- Einfacheres Arbeiten lokal

# Trick #4: Mocking Services

[www.wiremock.org](http://www.wiremock.org)

[www.mbtest.org](http://www.mbtest.org) (Node.js)

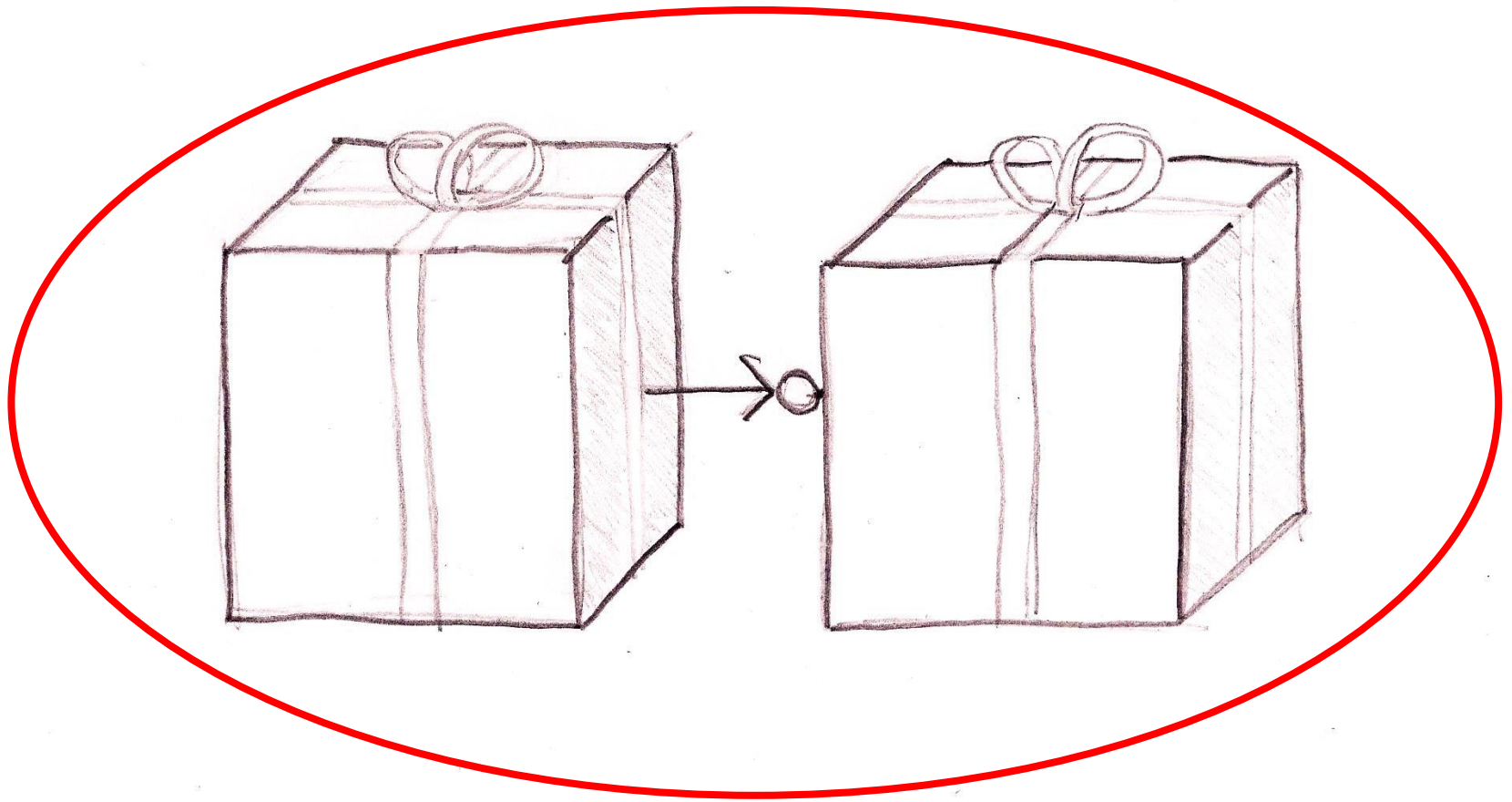
[github.com/dreamhead/moco](https://github.com/dreamhead/moco)

[github.com/azagniotov/stubby4j](https://github.com/azagniotov/stubby4j)

[github.com/vcr/vcr](https://github.com/vcr/vcr) (RUBY)

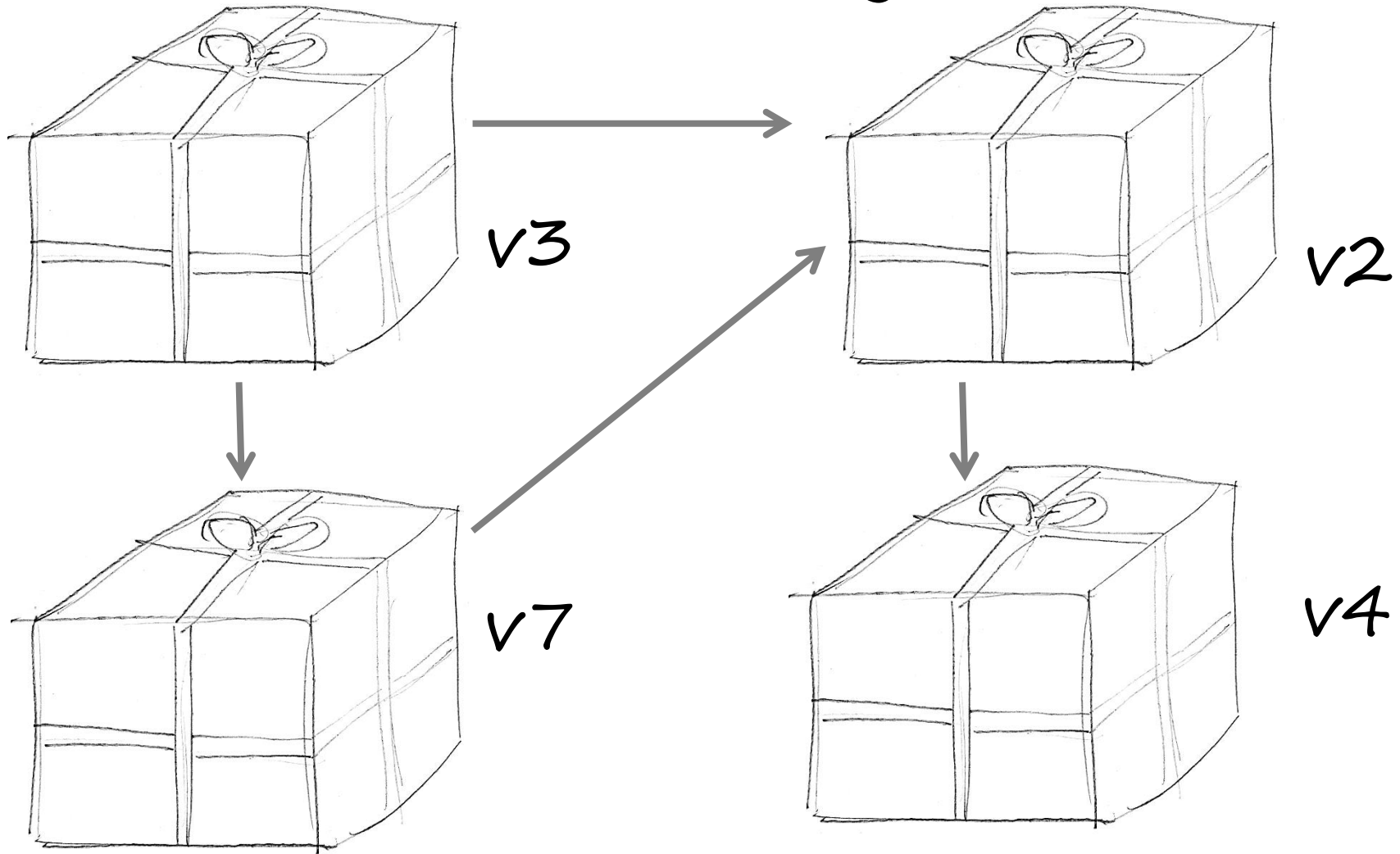
#5

Doppelt hält besser!

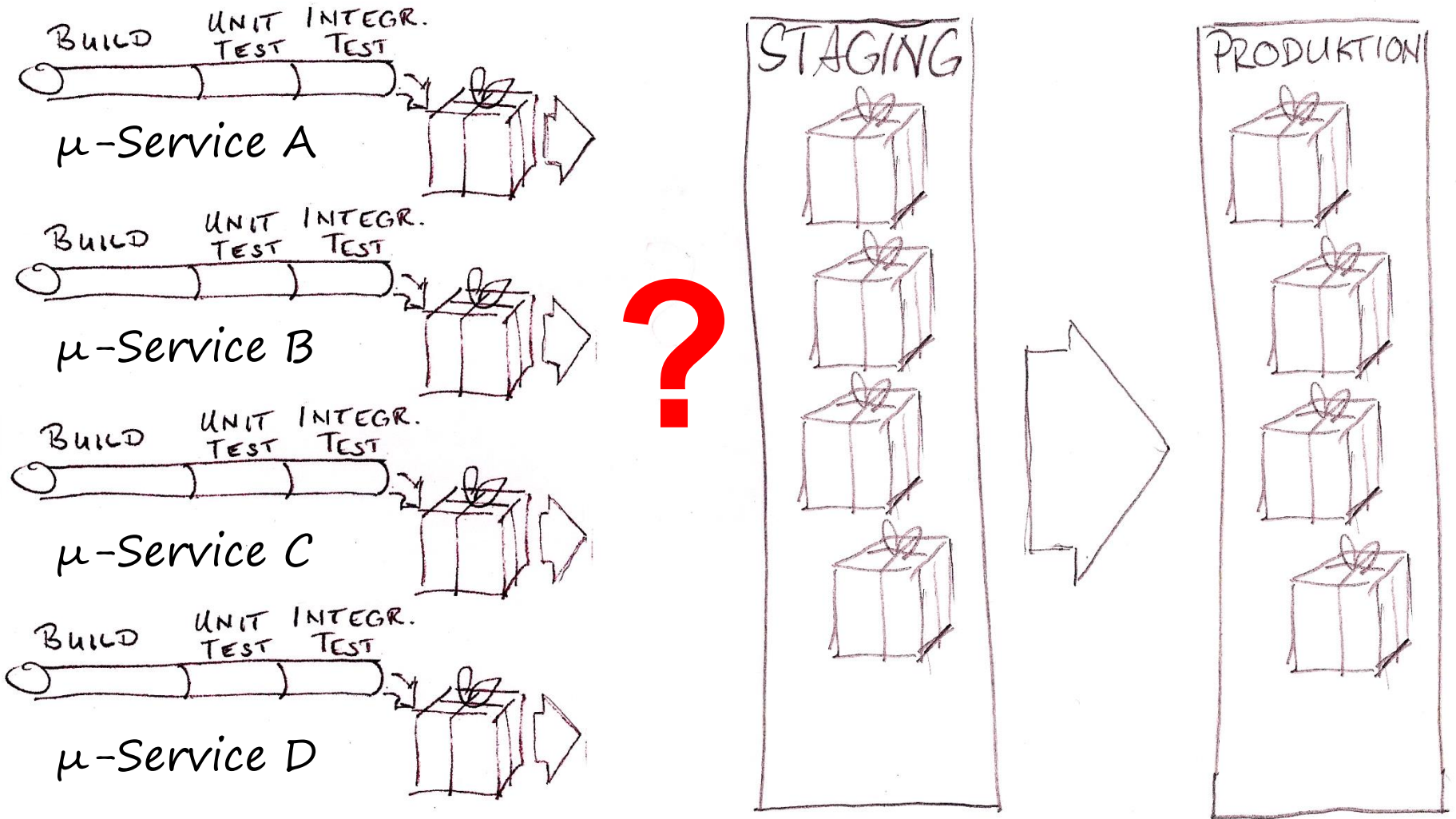




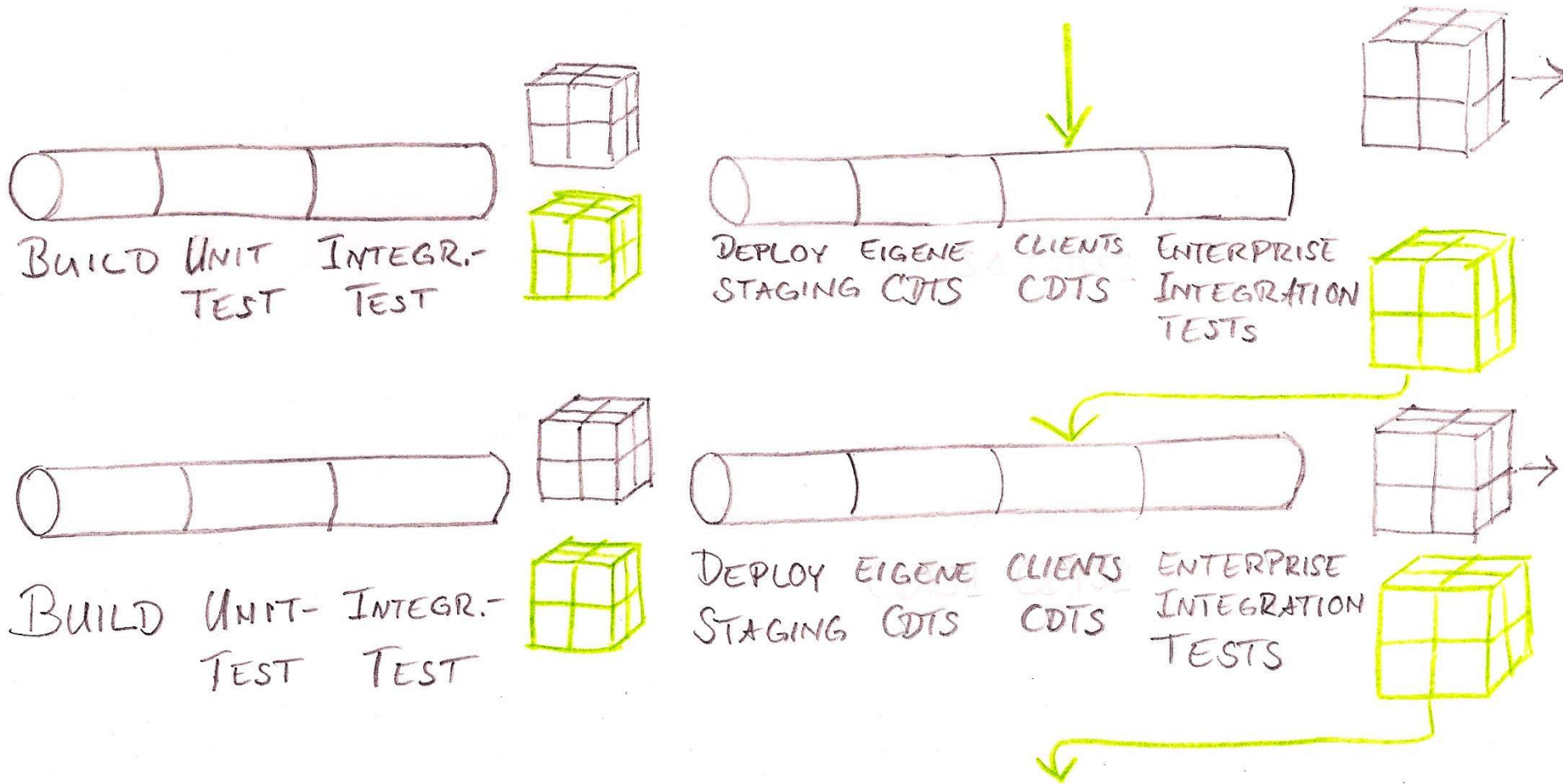
# Problem #5: Versionierung & CD



# Problem #5: Versionierung & CD



# Trick #5: Doppelt hält besser!

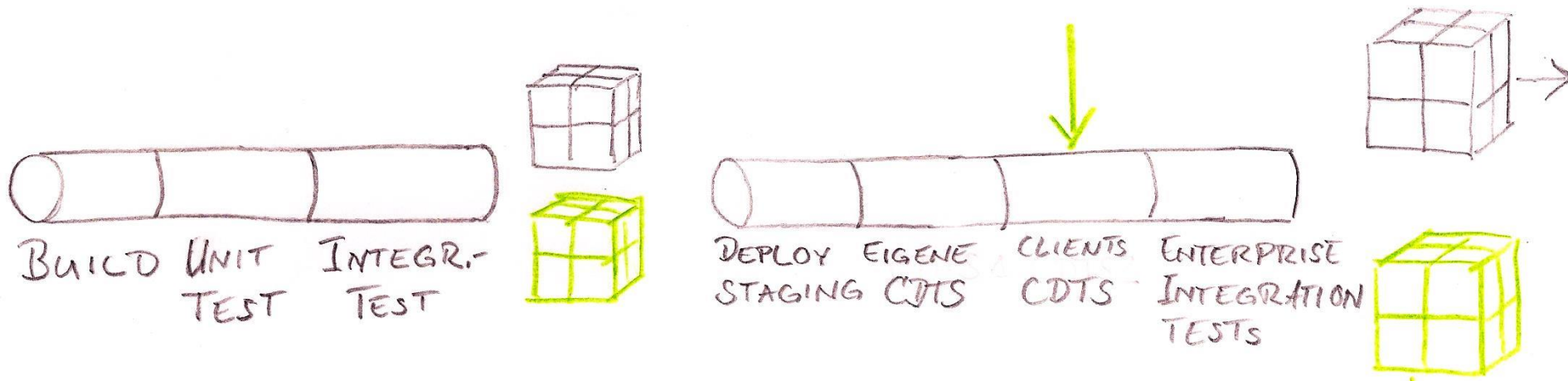


# Nur zwei Versionen

1. Die Produktive

2. Die Neue

→ Kein Versionsmanagement



# Unbreak Changes

## 1. Altes Schema

```
{  
  „id“ : 9586729475,  
  „vorname“ : „Max“,  
  „nachname“ : „Mustermann“,  
  „name“ : „Max Mustermann“  
}
```

# Unbreak Changes

1. Altes Schema
2. Eine Zwischenversion:  
altes UND neues Schema

```
{  
  „id“ : 9586729475,  
  „vorname“ : „Max“,  
  „nachname“ : „Mustermann“,  
  „name“ : „Max Mustermann“,  
  „displayName“ : „Max Mustermann“  
}
```

# Unbreak Changes

1. Altes Schema
2. Eine Zwischenversion:  
altes UND neues Schema
3. Altes Schema ausbauen

```
{  
  „id“ : 9586729475,  
  „vorname“ : „Max“,  
  „nachname“ : „Mustermann“,  
  „name“ : „Max Mustermann“,  
  „displayName“ : „Max Mustermann“  
}
```

# Trick #5: Doppelt hält besser

Wenn doch mal versehentlich zwei Services genau gleichzeitig ausrollen, die nicht miteinander arbeiten können?



# Trick #5: Doppelt hält besser

- Small Changesets

# Trick #5: Doppelt hält besser

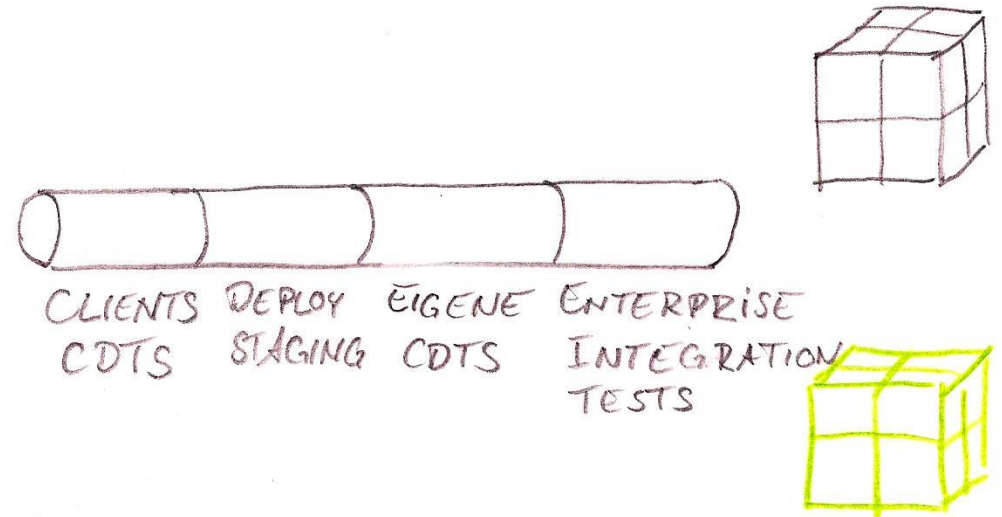
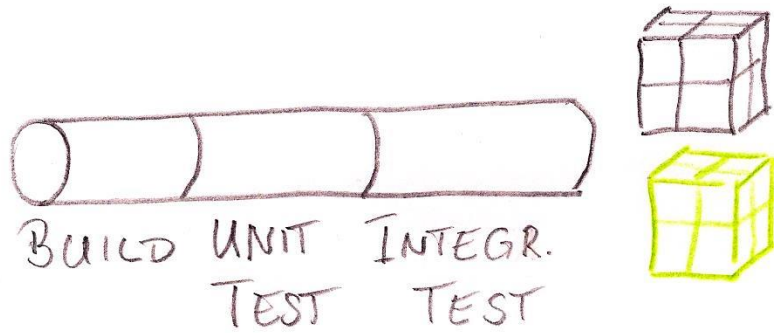
- Small Changesets
- Echtes Continuous Deployment

# Trick #5: Doppelt hält besser

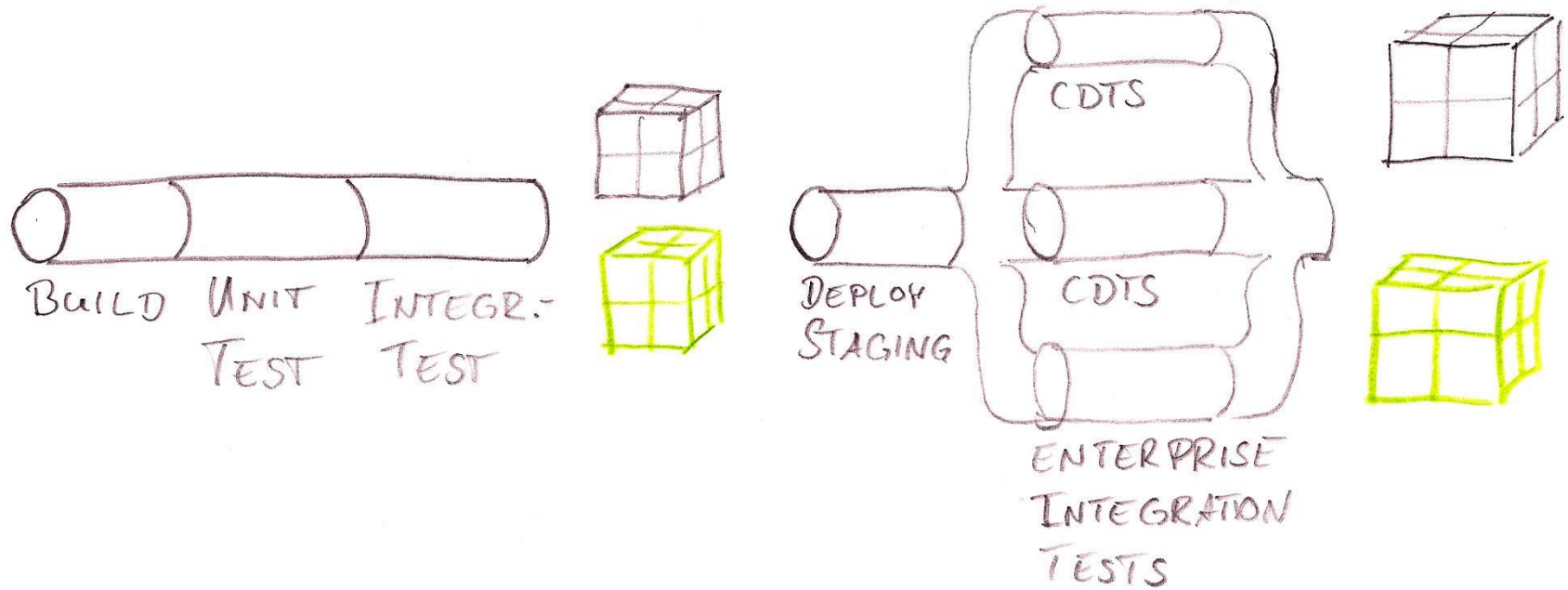
- Small Changesets
- Echtes Continuous Deployment
- Miteinander Reden!

# Trick #5: Doppelt hält besser

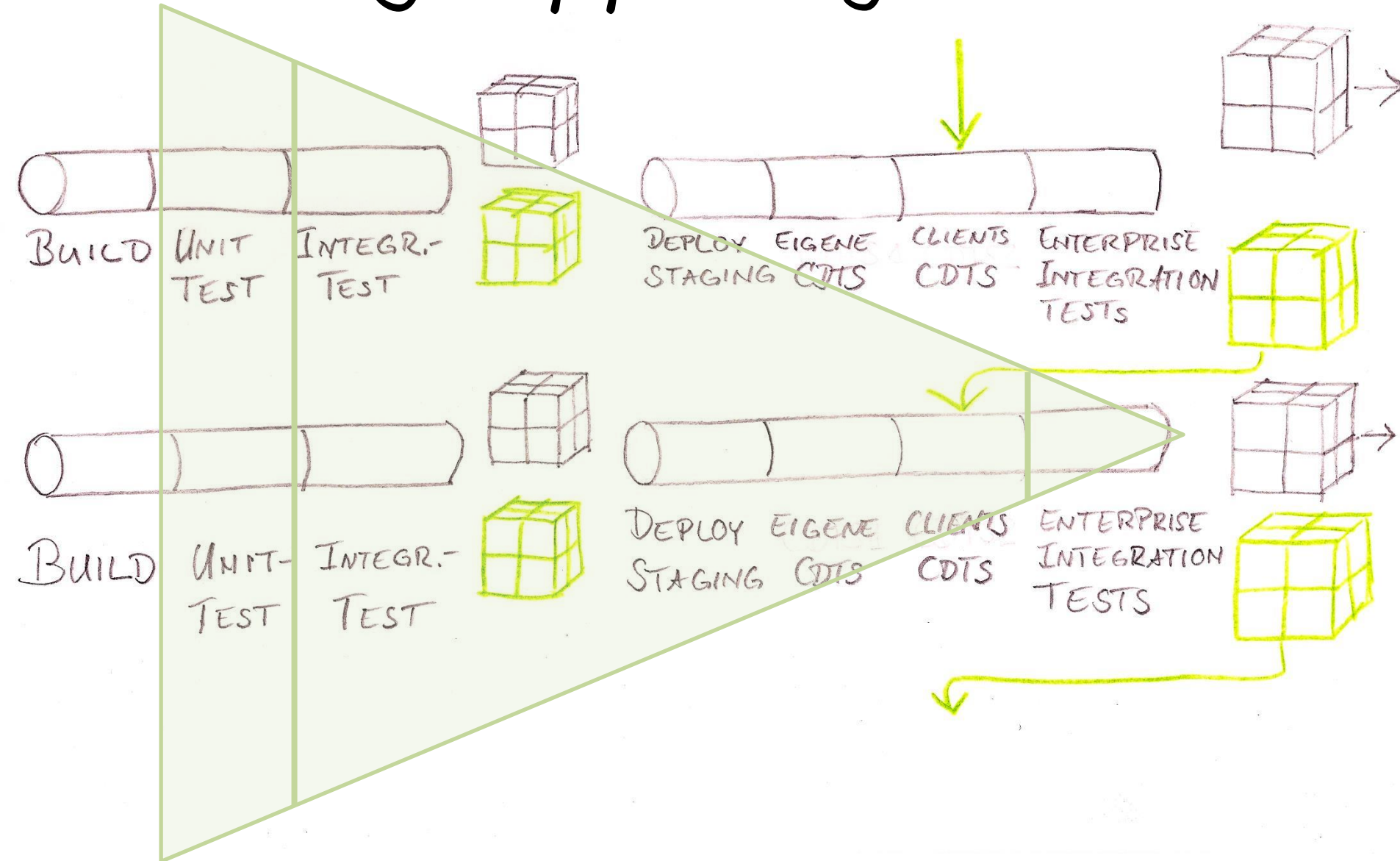
## - Variante 1



# Trick #5: Doppelt hält besser - Variante 2



# Die gekippte Pyramide



# Tipps & Tricks

1. Postel's Law:  
Teste nicht zu detailliert
2. Drück' Dich aus:  
Abstrahiere mit Response-Objects
3. Production Code ist Test Code:  
Einfache Consumer Driven Tests
4. Täusche und Betrüge:  
Benutze Mocks für Downstream Dependencies
5. Doppelt hält besser: CDTS in der Deployment Pipeline

# Zum Weiterlesen

- Mike Cohn:  
<http://www.mountaingoatsoftware.com/blog/the-forgotten-layer-of-the-test-automation-pyramid>
- Ian Robinson: Consumer-Driven Contracts: A Service Evolution Pattern  
<http://martinfowler.com/articles/consumerDrivenContracts.html>
- Brandon Byars: Enterprise Integration Using REST  
<http://martinfowler.com/articles/enterpriseREST.html>
- Jay Fields: Working Effectively with Unit Tests  
<https://leanpub.com/wewut>



# Tipps & Tricks

1. Postel's Law:  
Teste nicht zu detailliert
2. Drück' Dich aus:  
Abstrahiere mit Response-Objects
3. Production Code ist Test Code:  
Einfache Consumer Driven Tests
4. Täusche und Betrüge:  
Benutze Mocks für Downstream Dependencies
5. Doppelt hält besser: CDTS in der Deployment Pipeline