

API-Definition mal anders

Schnittstellen aus Sicht des Clients definieren



<https://docs.PACT.io>



Werner Eberling (@wer_eb)
werner.eberling@mathema.de

www.mathema.de

Der Sprecher



Werner Eberling

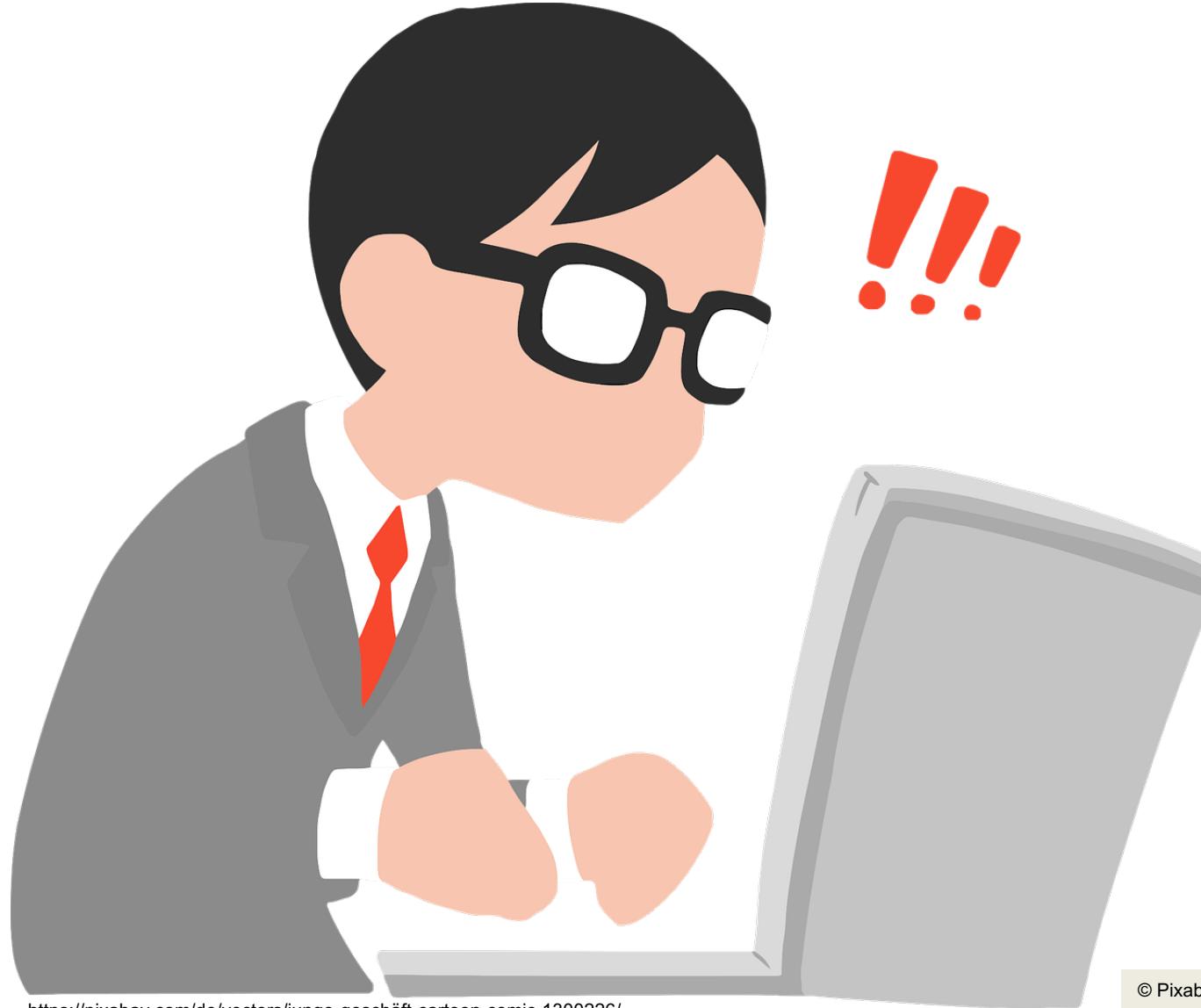
Principal Consultant / Autor

Email: werner.eberling@mathema.de

Twitter: [@Wer_Eb](https://twitter.com/Wer_Eb)



OMG was ist den das für eine Schnittstelle?



<https://pixabay.com/de/vectors/junge-geschäft-cartoon-comic-1300226/>

© Pixabay Licence

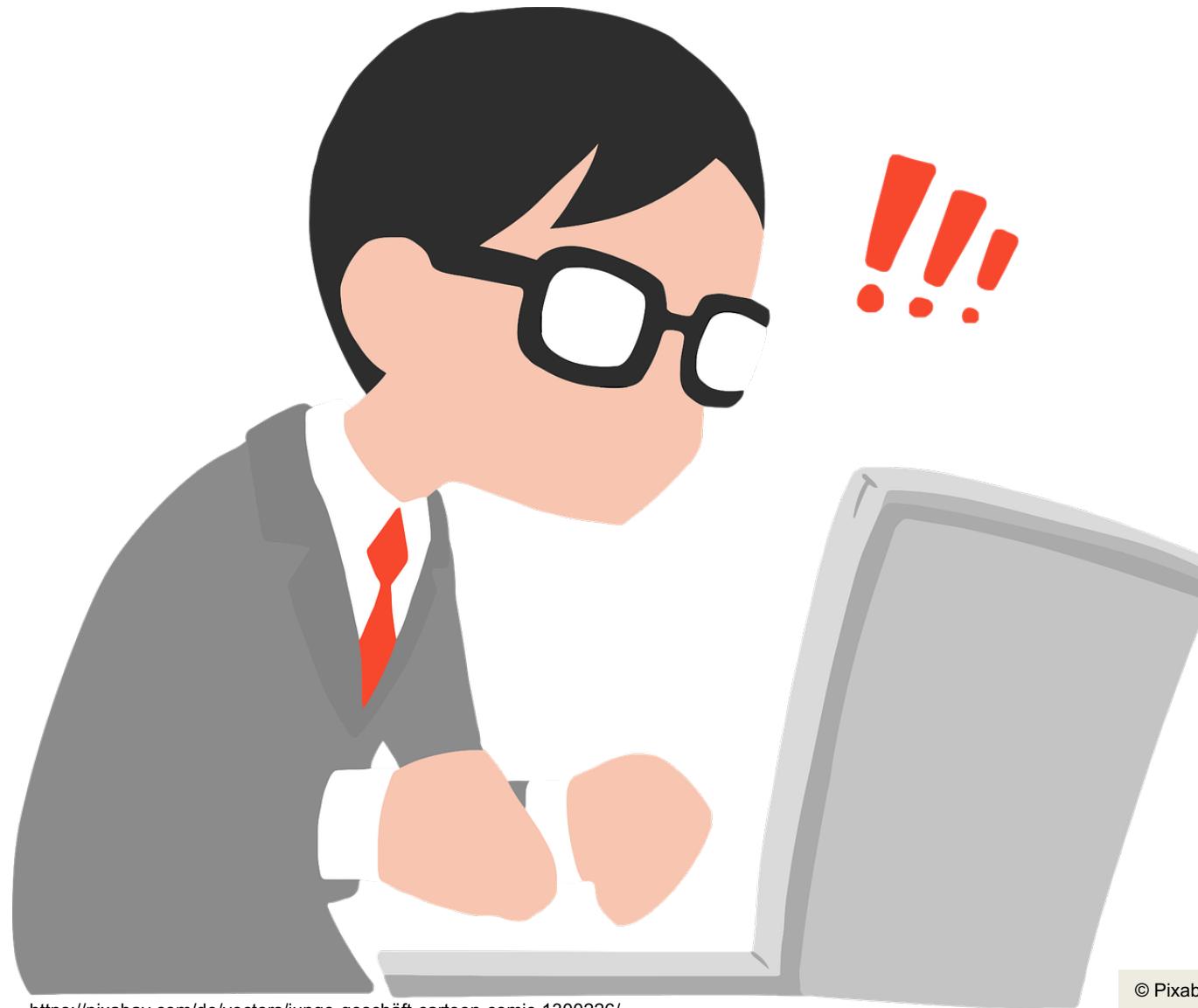
Eine kurze Geschichte zum Thema



Wie wäre es den Client zu fragen?



Das war aber anders abgesprochen!!!



<https://pixabay.com/de/vectors/junge-geschäft-cartoon-comic-1300226/>

© Pixabay Licence

Lasst uns einen Vertrag machen



Design by Contract - Das Ende der Flexibilität?



Postels Gesetz

„sei streng bei dem was du tust und
offen bei dem was du von anderen akzeptierst“

– RFC 761

Clients als Tolerant Reader

- Der Client definiert was er macht und braucht...

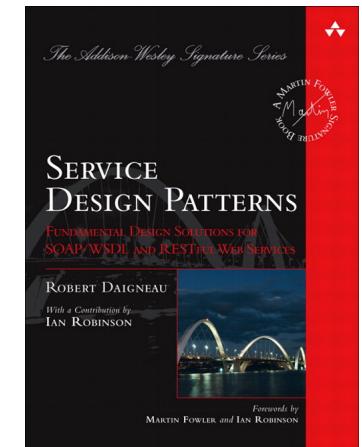


© Pixabay Licence

- ... und ignoriert alles andere.



© Pixabay Licence



<https://martinfowler.com/bliki/TolerantReader.html>

Wie war das jetzt mit dem Vertrag?



Testen – und zwar alles!!!



Logisch, oder nicht?

Moment, wie ist das im echten Leben?



Gleich das ganze Haus anzünden?

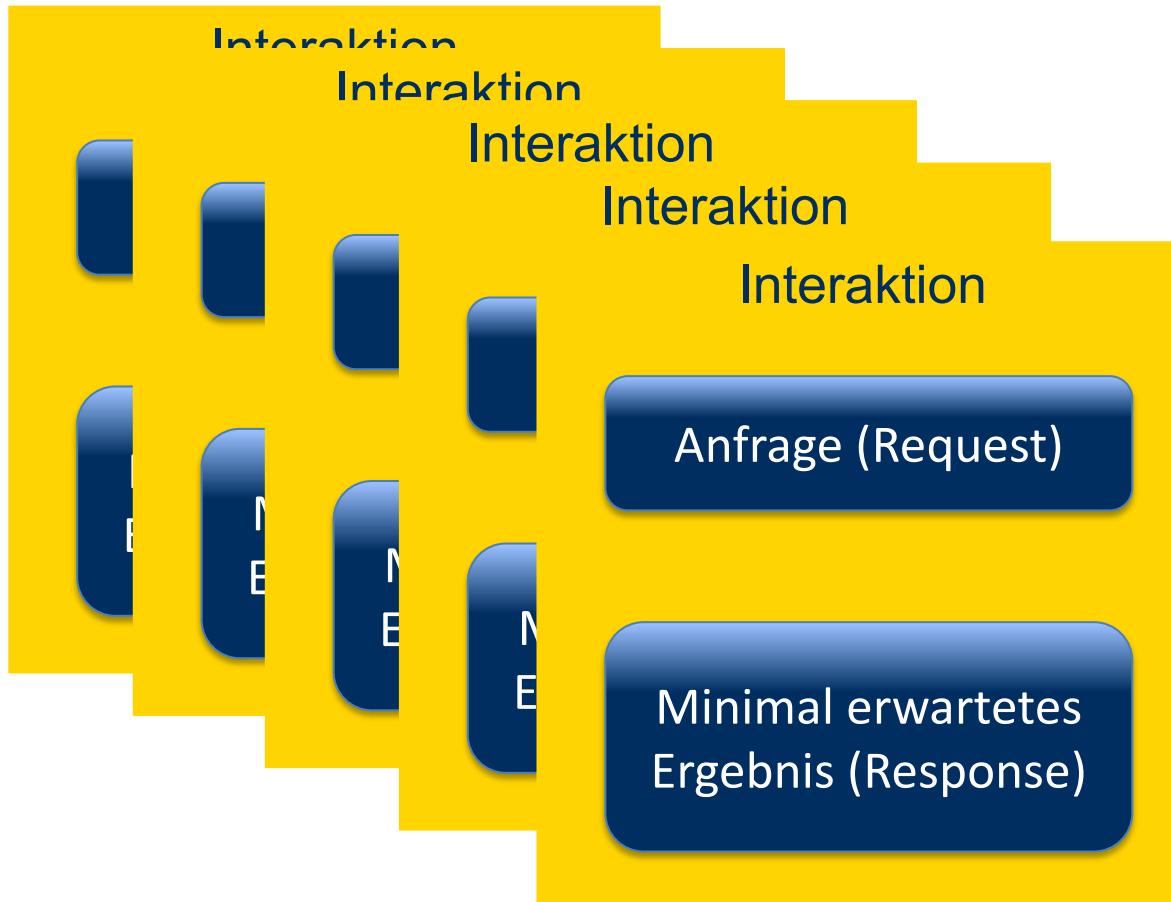


Wohl eher den Signaleingang simulieren...



Interaktionen statt starrer Schnittstellen

Beschreibung der Schnittstelle aus Sicht des Clients



Contract Testing

PACTS

<https://docs.PACT.io>

VS.



Spring Cloud Contract

<https://cloud.spring.io/spring-cloud-contract>

Contract Testing mit PACT

PACT (noun):

A formal agreement between individuals or parties.

"The country negotiated a trade PACT with the US"

Synonyms: agreement, protocol, deal, contract

~ Oxford Dictionaries

(von: <https://docs.PACT.io>)



<https://docs.PACT.io>

Contract Testing mit PACT – PACT-jvm

The screenshot shows a GitHub README.md page for the **pact-jvm** project. The page is displayed in a web browser window with a light gray header bar containing standard OS X-style icons (red, yellow, green circles, back, forward, etc.) and a URL field showing "github.com". The main content area has a white background with a dark gray header bar labeled "README.md".

pact-jvm

build passing build failing maven central 3.6.6

JVM implementation of the consumer driven contract library [pact](#).

From the [Ruby Pact website](#):

Define a pact between service consumers and providers, enabling "consumer driven contract" testing.

Pact provides an RSpec DSL for service consumers to define the HTTP requests they will make to a service provider and the HTTP responses they expect back. These expectations are used in the consumers specs to provide a mock service provider. The interactions are recorded, and played back in the service provider specs to ensure the service provider actually does provide the response the consumer expects.

This allows testing of both sides of an integration point using fast unit tests.

This gem is inspired by the concept of "Consumer driven contracts". See <http://martinfowler.com/articles/consumerDrivenContracts.html> for more information.

Read [Getting started with Pact](#) for more information on how to get going.

Contact

- Twitter: [@pact_up](#)
- Slack: [Join the chat at http://slack.pact.io/](http://slack.pact.io/)
- Stack Overflow: <https://stackoverflow.com/questions/tagged/pact>

Wie sieht so ein Vertrag aus?

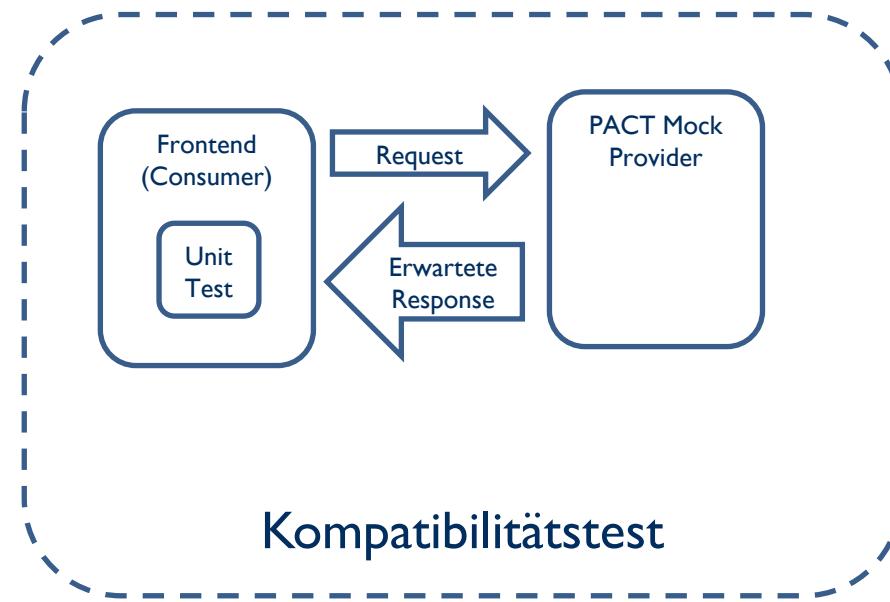
```
{  
  "provider": {  
    "name": "HelloWorldProvider"  
  },  
  "consumer": {  
    "name": "HelloWorldConsumer"  
  },  
  "interactions": [  
    {  
      "description": "Abfrage von HelloWorld",  
      "request": {  
        "method": "GET",  
        "path": "/"  
      },  
      "response": {  
        "status": 200,  
        "body": {  
          "response": "Hello World"  
        }  
      }  
    }  
  ],  
  "metadata": {  
    "PACTSpecification": {  
      "version": "3.0.0"  
    },  
    "PACT-jvm": {  
      "version": "3.6.6"  
    }  
  }  
}
```

Bonusfeature: PACTs als Mockup für Integrationstests

```
cdc-sample — pact-stub-service ./pacts/GreetingConsumer-GreetingProvider.json -p 9090 -h localhost — 124x23
Werners-MBP:cdc-sample wreberli$ pact-stub-service ./pacts/GreetingConsumer-GreetingProvider.json -p 9090 -h localhost
INFO: Loading interactions from ./pacts/GreetingConsumer-GreetingProvider.json
I, [2019-05-08T15:35:28.019161 #58468] INFO -- : Registered expected interaction GET /campus/greeting
D, [2019-05-08T15:35:28.019161 #58468] DEBUG -- : {
  "description": "GreetingConsumer-GreetingProvider"
}
I, [2019-05-08T15:36:25.017595 #58468] INFO -- : Received request GET /campus/greeting
D, [2019-05-08T15:36:25.017744 #58468] DEBUG -- : {
  "request": {
    "method": "GET",
    "uri": "/campus/greeting"
  }
}
INFO: WEBrick 1.4.2
INFO: ruby 2.3.7 (2018-03-28) [universal.x86_64-darwin17]
INFO: WEBrick::HTTPServer#start: pid=58468 port=9090
I, [2019-05-08T15:36:25.017595 #58468] INFO -- : Received request GET /campus/greeting
D, [2019-05-08T15:36:25.017744 #58468] DEBUG -- : {
  "request": {
    "method": "GET",
    "uri": "/campus/greeting",
    "headers": {
      "Host": "localhost:9090",
      "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
      "Upgrade-Insecure-Requests": "1",
      "Cookie": "_gid=GA1.1.1010395921.1557305790; _ga=GA1.1.1159609082.1557305790",
      "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/12.1 Safari/605.1.15",
      "Accept-Language": "de-de",
      "Accept-Encoding": "gzip, deflate",
      "Connection": "keep-alive",
      "Version": "HTTP/1.1"
    }
  }
}
I, [2019-05-08T15:36:25.017946 #58468] INFO -- : Found matching response for GET /campus/greeting
D, [2019-05-08T15:36:25.018067 #58468] DEBUG -- : {
  "status": 200,
  "headers": {},
  "body": {
    "greeting": "Hallo vom MATHEMA Campus!"
  }
}
```

Beispiel

Consumer prüft Kompatibilität gegen den PACT



PACT Erzeugung via UnitTest

```
@PACT(provider="GreetingProvider", consumer = "GreetingConsumer")
public RequestResponsePACT createPACT(PACTDslWithProvider builder) {
    return builder
        .uponReceiving("Abfrage aller Grüesse mit vorhandenen Daten")
            .path("/greetings")
            .method("GET")
        .willRespondWith()
            .status(200)
        .body(new PACTDslJsonBody()
            .array("greetings")
            .object()
                .stringValue("type", "formal")
                .stringValue("phrase", "Good morning")
                .closeObject()
            .object()
                .stringValue("type", "casual")
                .stringValue("phrase", "Hey")
                .closeObject()
        )
    .toPACT();
}
```

Consumer prüft seine eigene Kompatibilität im selben Test

```
@ExtendWith(PACTConsumerTestExt.class)
@PACTTestFor(providerName = "GreetingProvider")
@PACTFolder("PACTs")
public class GreetingConsumerTest
{
    @PACT(provider="GreetingProvider", consumer="GreetingConsumer")
    public RequestResponsePACT createPACT(PACTDslWithProvider builder) {
        return builder
            .uponReceiving("Abfrage aller Grüesse ohne vorhandene Daten")
            .path("/greetings")
            .method("GET")
            .willRespondWith()
            .status(200)
            .body(new PACTDslJsonBody().array("greetings"))    }

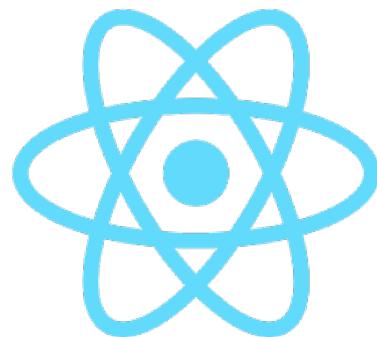
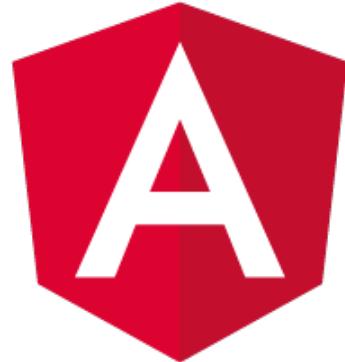
    @Test
    void testAbfrageAllerGruesse(MockServer mockServer) throws Exception {
        HttpResponse resp = Request.Get(mockServer.getUrl() + "/greetings").execute().returnResponse();
        Assertions.assertEquals(200, resp.getStatusLine().getStatusCode());
    }
}
```

Im “echten“ Projekt immer auch die “echten“
Clientklassen verwenden, anstatt mit dem
HttpClient zu testen!

Beispiel

```
<!-- Navbar -->
<div class="w3-top">
  <div class="w3-bar w3-black w3-hide-large w3-right" href="javascript:void(0)" title="Toggle Navigation Menu"><i class="fa fa-bars"></i> HOME</a>
  <a href="#" class="w3-bar-item w3-button w3-padding-large w3-hide-small w3-right" href="javascript:void(0)">BAND</a>
  <a href="#tour" class="w3-bar-item w3-button w3-padding-large w3-hide-small w3-right" href="javascript:void(0)">TOUR</a>
  <a href="#contact" class="w3-bar-item w3-button w3-padding-large w3-hide-small w3-right" href="javascript:void(0)">CONTACT</a>
<div class="w3-dropdown-hover w3-hide-small w3-right">
  <button class="w3-padding-large w3-button" title="More" href="javascript:void(0)"><i class="fa fa-caret-down"></i> MORE <i class="w3-large w3-hide-small w3-right" href="javascript:void(0)"> </i></button>
  <div class="w3-dropdown-content w3-bar-block w3-card-4 w3-right">
    <a href="#" class="w3-bar-item w3-button w3-right" href="javascript:void(0)">Merchandise</a>
    <a href="#" class="w3-bar-item w3-button w3-right" href="javascript:void(0)">Extras</a>
    <a href="#" class="w3-bar-item w3-button w3-right" href="javascript:void(0)">Media</a>
  </div>
</div>
<a href="javascript:void(0)" class="w3-hide-small w3-right" href="javascript:void(0)"> </a>
</div>
</div>
```

Aber: UIs entstehen heutzutage in JavaScript oder TypeScript



PACT bauen und Verifizieren im Karma Test

```
describe('create()', () => {
  beforeAll((done) => {
    provider.addInteraction({
      uponReceiving: 'Abfrage aller Grüsse ohne vorhandene Daten',
      withRequest: {
        method: 'GET',
        path: '/greetings'
      },
      willRespondWith: {
        status: 200,
        body: {
          greetings: []
        }
      }
    }).then(done, error => done.fail(error));
  });

  it('should return no Greetings', (done) => {
    const userService: GreetingService = TestBed.get(GreetingService);
    userService.loadGreetings().subscribe(response => {
      expect(response).toEqual({greetings: []});
      done();
    }, error => {
      done.fail(error);
    });
  });
});
```

Mock Server im Karma-Test konfigurieren und einbinden

```
beforeAll(done => {
  provider = new PACTWeb({
    consumer: 'GreetingConsumer',
    provider: 'GreetingProvider',
    port: 1234,
    host: '127.0.0.1',
    log: path.resolve(process.cwd(), 'logs', 'PACT.log'),
  });
  // required for slower CI environments
  setTimeout(done, 2000);
  // Required if run with `singleRun: false`
  provider.removeInteractions();
});
```

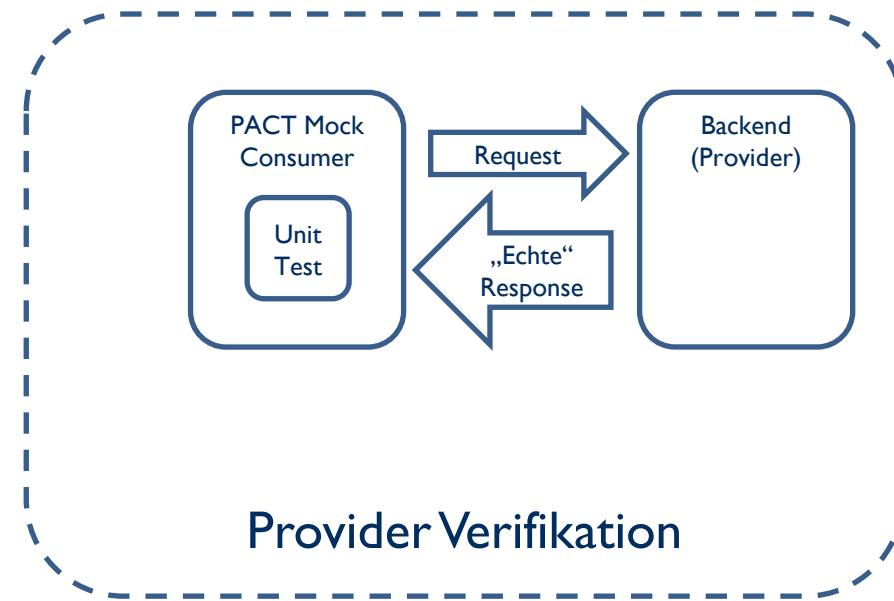
greeting.service.PACT.spec.ts

karma.conf.js

```
module.exports = function (config) {
  config.set({
    ...
    pact: [
      {
        cors: true,
        port: 8080,
        consumer: "GreetingConsumer",
        provider: "GreetingProvider",
        dir: "pacts",
        spec: 3
      }
    ]
  });
};
```

Beispiel

Server validiert Verhalten gegen den PACT (I)



Provider verifiziert sein Verhalten gegen den PACT

```
@ExtendWith(SpringExtension.class)
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@Provider("GreetingProvider")
@PACTUrl(urls = "PACTs/GreetingConsumer-GreetingProvider.json")
public class GreetingProviderVerificationTest {

    @LocalServerPort
    int randomServerPort;

    @Autowired
    GreetingService greetingService;

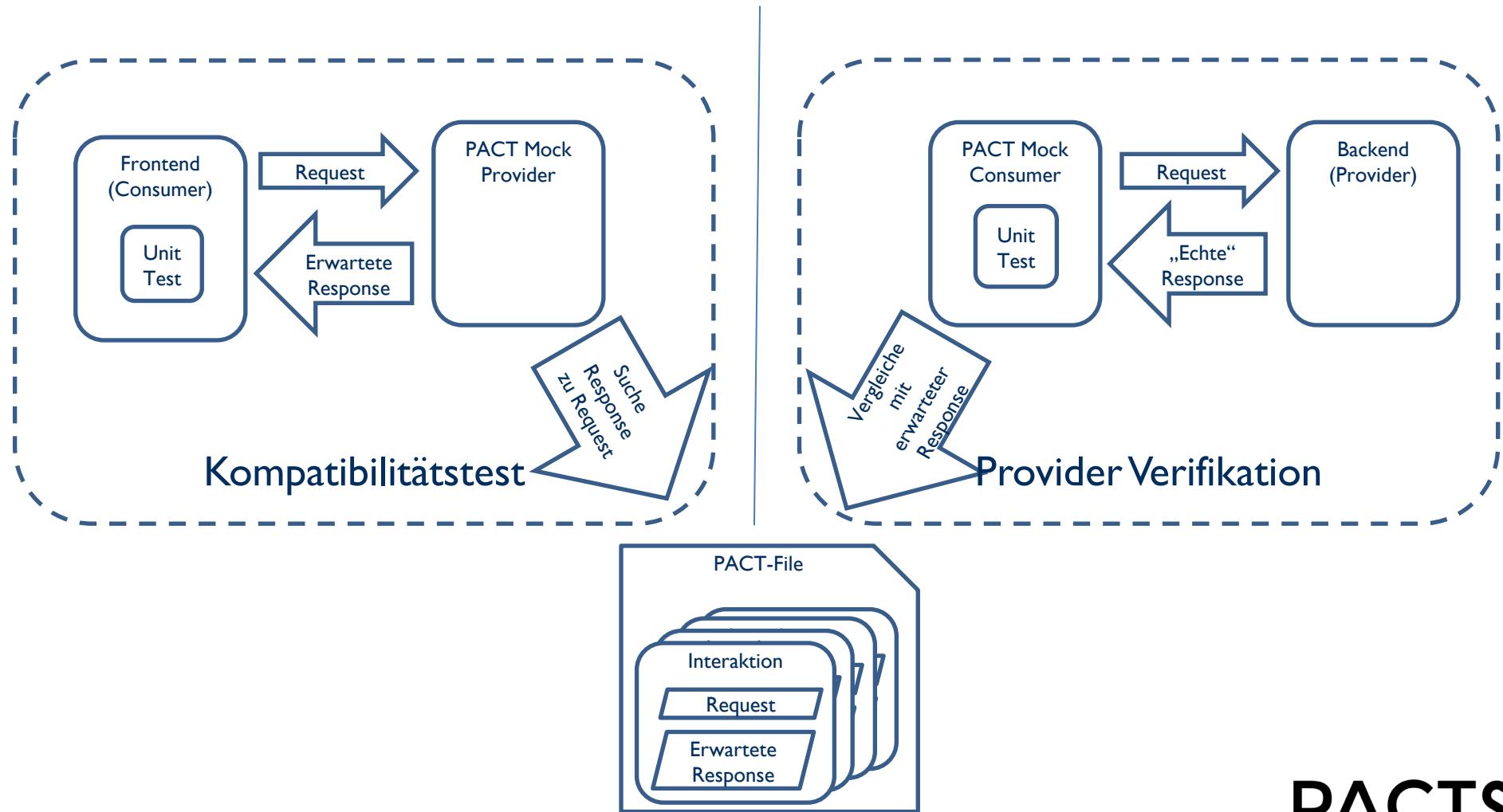
    @TestTemplate
    @ExtendWith(PACTVerificationInvocationContextProvider.class)
    void PACTVerificationTestTemplate(PACTVerificationContext context) {
        context.verifyInteraction();
    }

    @BeforeEach
    void before(PACTVerificationContext context) throws Exception {
        context.setTarget(HttpTestTarget.fromUrl(new URL("http", "localhost", randomServerPort, "")));
        greetingService.reset();
    }
}
```

Beispiel

```
<!-- Navbar -->
<div class="w3-top">
  <div class="w3-bar w3-black w3-hide-large w3-right" href="javascript:void(0)" title="Toggle Navigation Menu"><i class="fa fa-bars"></i> HOME</a>
  <a href="#" class="w3-bar-item w3-button w3-padding-large w3-hide-small w3-right" href="javascript:void(0)">BAND</a>
  <a href="#tour" class="w3-bar-item w3-button w3-padding-large w3-hide-small w3-right" href="javascript:void(0)">TOUR</a>
  <a href="#contact" class="w3-bar-item w3-button w3-padding-large w3-hide-small w3-right" href="javascript:void(0)">CONTACT</a>
<div class="w3-dropdown-hover w3-hide-small w3-right">
  <button class="w3-padding-large w3-button" title="More" href="javascript:void(0)"><i class="fa fa-caret-down"></i> MORE <i class="w3-large w3-hide-small w3-right" href="javascript:void(0)"> </i></button>
  <div class="w3-dropdown-content w3-bar-block w3-card-4 w3-right">
    <a href="#" class="w3-bar-item w3-button w3-right" href="javascript:void(0)">Merchandise</a>
    <a href="#" class="w3-bar-item w3-button w3-right" href="javascript:void(0)">Extras</a>
    <a href="#" class="w3-bar-item w3-button w3-right" href="javascript:void(0)">Media</a>
  </div>
</div>
<a href="javascript:void(0)" class="w3-hide-small w3-right" href="javascript:void(0)"> </a>
</div>
</div>
```

Verträge automatisiert & unabhängig verifizieren



PACT
<https://docs.PACT.io>

Entwicklung mit Consumer Driven Contracts

- Entkopplung von Frontend- und Service-Entwicklung durch den Einsatz von Consumer Driven Contracts
 - Definition der Interaktionen zwischen Frontend und Service in Form von PACT Files
 - Test und Entwicklung des Frontends gegen aus diesem PACT File erzeugte MockUps bzw. Kompatibilitätstests
 - Entwicklung und Verifikation der Services gegen aus PACT File erzeugte Verifikationstests
- Automatisierte Integrationstests „ohne zu integrieren“



PACT basierte Integrationstests

- Gut für unabhängige und schnelle Entwicklung
 - Einfache Mockups „for free“
- ABER:
 - Zustandsänderungen NICHT abbildbar!
 - Nicht geeignet für komplexe UI Flows
 - Abwesenheit von Elementen nicht testbar



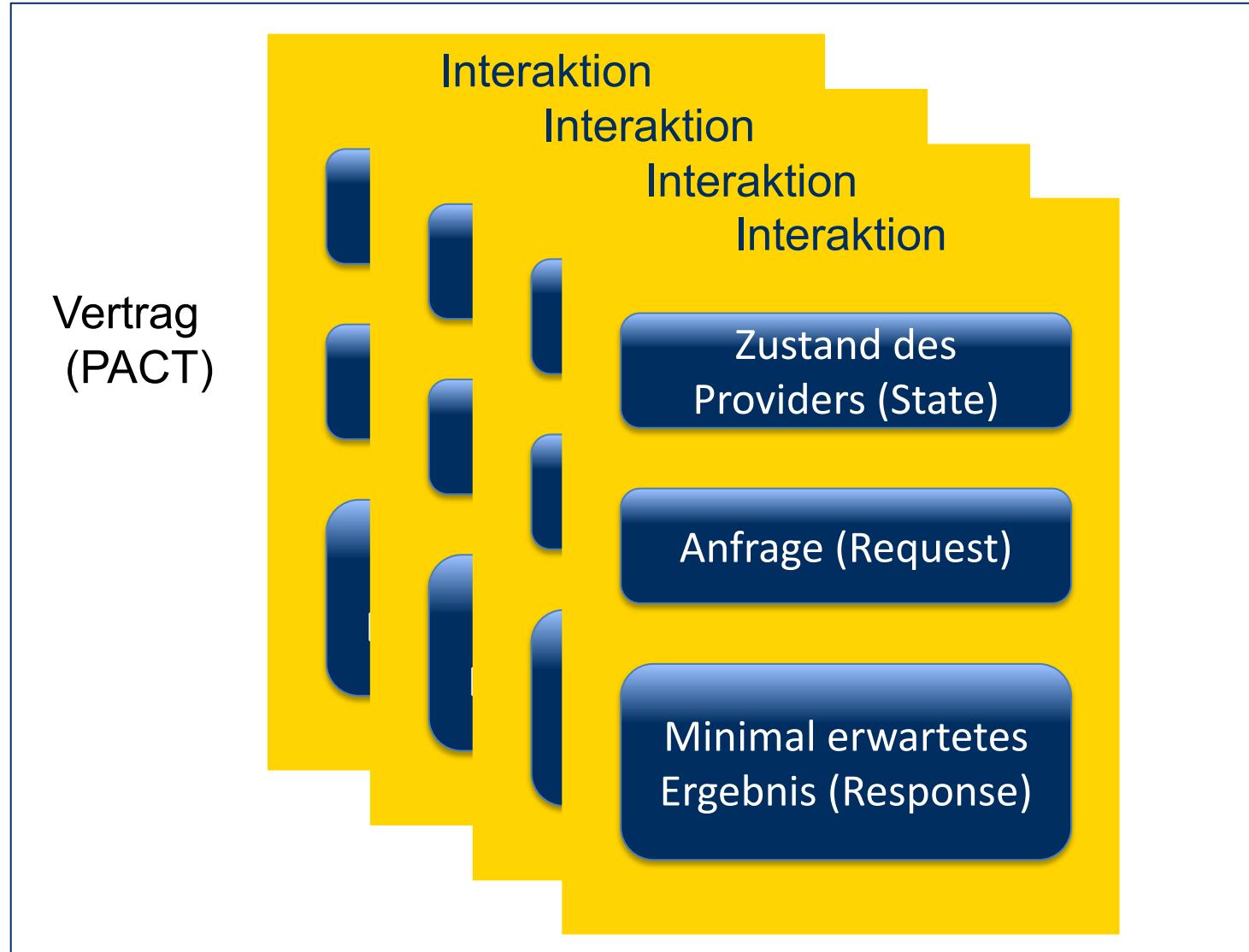
<https://pixabay.com/de/photos/schraubendreher-hintergrund-schraube-1008974/>

© CC0 Public Domain

Zustände statt komplexer Abläufe (I)



Zustände statt komplexer Abläufe (II)



PACT Erzeugung mit State

```
@PACT(provider="GreetingProvider", consumer = "GreetingConsumer")
public RequestResponsePACT createPACT(PACTDslWithProvider builder) {
    return builder
        .given("Es gibt zwei Grüesse")
        .uponReceiving("Abfrage aller Grüesse mit vorhandenen Daten")
            .path("/greetings")
            .method("GET")
        .willRespondWith()
            .status(200)
        .body(new PACTDslJsonBody()
            .array("greetings")
            .object()
                .stringValue("type", "formal")
                .stringValue("phrase", "Good morning")
                .closeObject()
            .object()
                .stringValue("type", "casual")
                .stringValue("phrase", "Hey")
                .closeObject()
        )
        .toPACT();
}
```

Aber: Die ganzen Testdaten!!!!



© Pixabay Licence

<https://pixabay.com/de/illustrations/panik-gro%C3%9Fe-augen-krummen-arm-1393619>

Struktur vs. Inhalt

- Akzeptanztest prüfen **Logik und Inhalte**
- Kompatibilitätstests prüfen **Struktur**
- CDC stellt Kompatibilität sicher, nicht Akzeptanz!

Mit Matchern Struktur anstatt Inhalte verifizieren!

```
@PACT(provider="GreetingProvider", consumer = "GreetingConsumer")
public RequestResponsePACT createPACT(PACTDslWithProvider builder) {
    return builder
        .given("Es gibt zwei Grüesse")
        .uponReceiving("Abfrage aller Grüesse mit vorhandenen Daten")
            .path("/greetings")
            .method("GET")
        .willRespondWith()
            .status(200)
            .body(new PactDslJsonBody()
                .array("greetings")
                    .object()
                        .stringType("type")
                        .stringType("phrase")
                        .closeObject()
                    .object()
                        .stringType("type")
                        .stringType("phrase")
                        .closeObject()
                )
            .toPACT();
}
```

PACT File mit Type-Matchern und Provider-States

```
{  
  "description": "Abfrage eines existierenden Grusses",  
  "request": {  
    ...  
  },  
  "response": {  
    "status": 200,  
    "body": {  
      ...  
    },  
    "matchingRules": {  
      "body": {  
        "$.greeting.type": {  
          "matchers": [  
            {  
              "match": "type"  
            }  
          ],  
          "combine": "AND"  
        },  
        "$.greeting.phrase": {  
          "matchers": [  
            {  
              "match": "type"  
            }  
          ],  
          "combine": "AND"  
        }  
      }  
    },  
    "providerStates": [  
      {  
        "name": "Es gibt einen casual Gruss"  
      }  
    ]  
  }  
}
```

Beispiel

PACTs Verwalten via PACT Broker

The screenshot shows a web browser window titled "Pacts" at the URL "localhost:9080". The page displays a table of pacts with the following columns: Consumer (with a dropdown arrow icon), Provider (with a file/folder icon), Latest pact published, Webhook status, and Last verified. There are two rows of data:

| Consumer ↕ | Provider ↕ | Latest pact published | Webhook status | Last verified | ... |
|------------------|------------------|-----------------------|----------------|---------------|-----|
| Example App | Example API | 1 day ago | Create | 1 day ago | ... |
| GreetingConsumer | GreetingProvider | 5 minutes ago | Create | | ... |

Below the table, a message indicates "2 pacts".

PACT File in den Broker laden



```
Werners-MacBook-Pro:~ wreberli$ curl -v -XPUT \-H "Content-Type: application/json" \  
> -d@pacts/GreetingConsumer-GreetingProvider.json \  
> http://localhost/pacts/provider/GreetingProvider/consumer/GreetingConsumer/version/1.0.0
```

Gegen ein PACT File aus dem Broker verifizieren

```
@ExtendWith(SpringExtension.class)
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@Provider("GreetingProvider")
@PactBroker(host="localhost", port="9080")
public class GreetingProviderVerificationTest {

    @LocalServerPort
    int randomServerPort;

    @Autowired
    GreetingService greetingService;

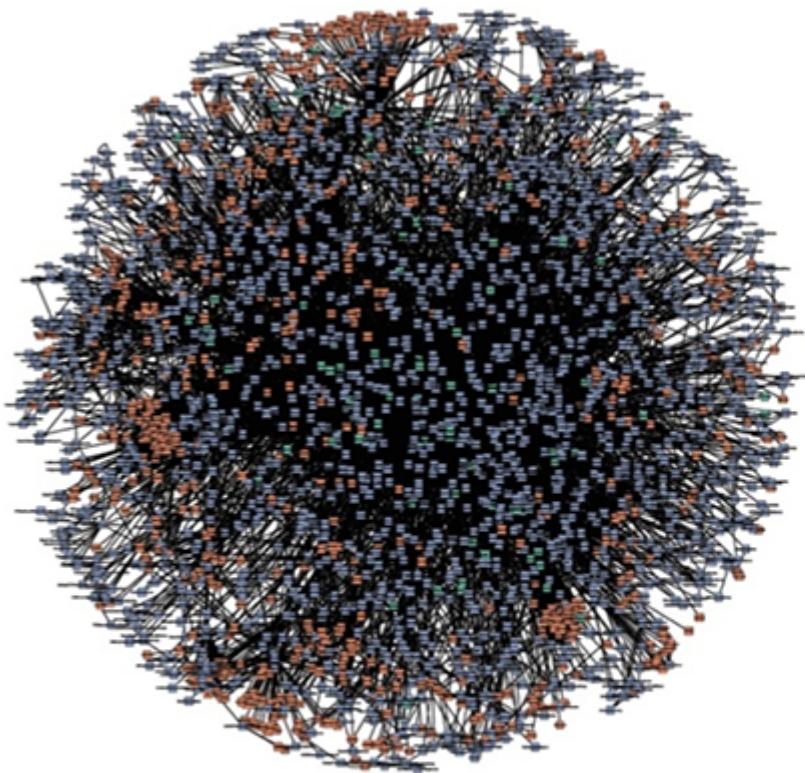
    @TestTemplate
    @ExtendWith(PACTVerificationInvocationContextProvider.class)
    void PACTVerificationTestTemplate(PACTVerificationContext context) {
        System.setProperty("PACT.provider.version", "1.0.0");
        System.setProperty("PACT.verifier.publishResults", "true");
        context.verifyInteraction();
    }

    @BeforeEach
    void before(PACTVerificationContext context) throws Exception {
        context.setTarget(HttpTestTarget.fromUrl(new URL("http", "localhost", randomServerPort, "")));
        greetingService.reset();
    }
}
```

Beispiel

Vertrage auch bei Inter-Service-Kommunikation (I)

■ Welcome to Deathstar Architecture



amazon.com



NETFLIX

<https://www.appcentrica.com/the-rise-of-microservices>

Verträge auch bei Inter-Service-Kommunikation (I)

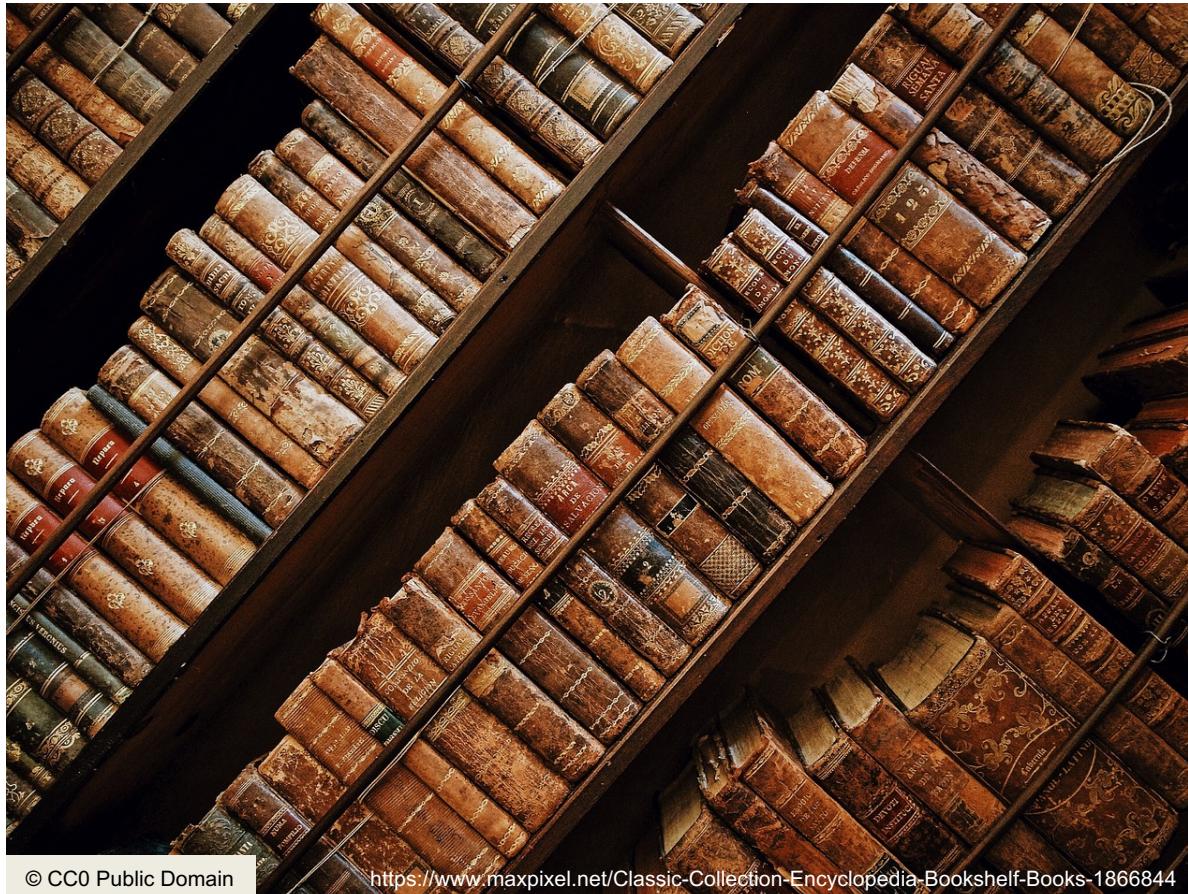
- Kommunikation zwischen Microservices ist eine Consumer – Provider Beziehung
- Über PACT Interaktionen kann ein Provider kompatible Änderungen absichern
- Über den PACT Broker kann ein Provider alle aktuell gültigen Verträge ermitteln ohne die Consumer direkt zu kennen

Aber wir haben doch schon Swagger?!?



Aber wir haben doch schon Swagger?!?

- Swagger beschreibt wie eine existierende Schnittstelle aussieht (Dokumentation)

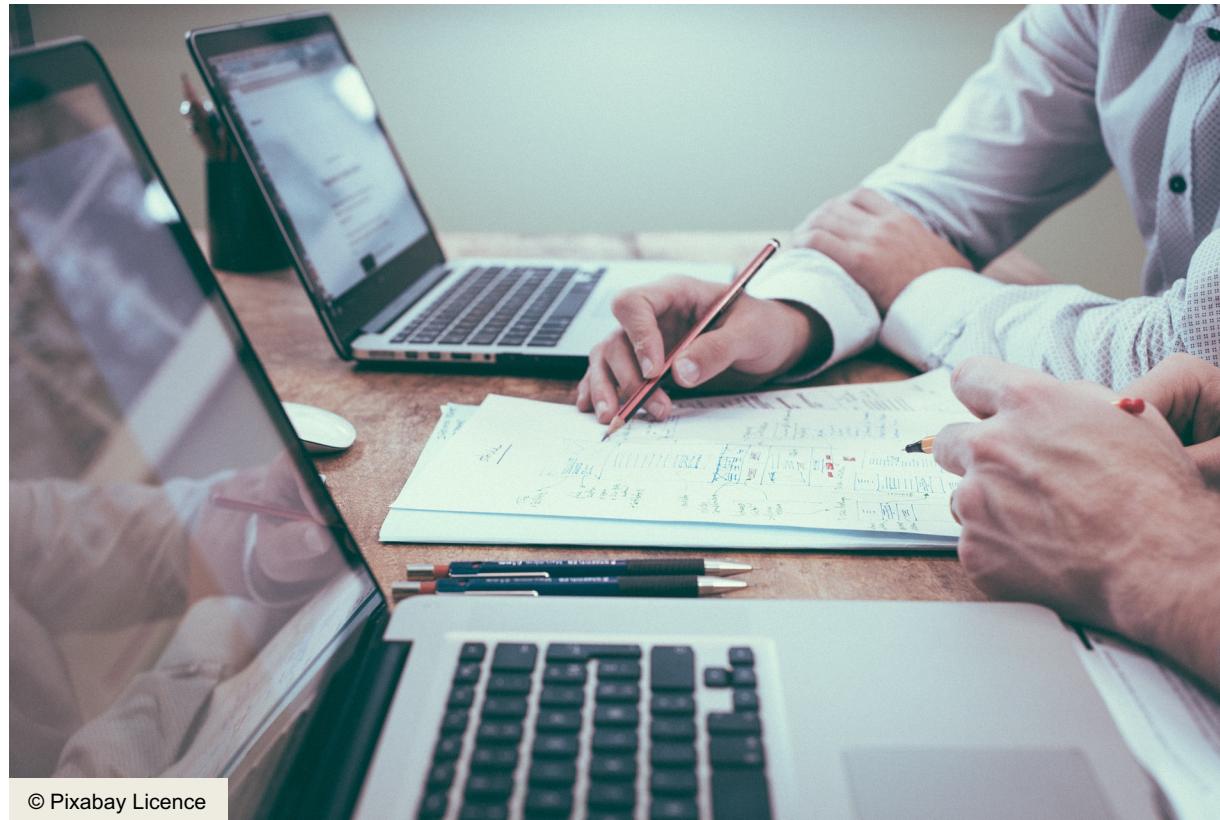


© CC0 Public Domain

<https://www.maxpixel.net/Classic-Collection-Encyclopedia-Bookshelf-Books-1866844>

Aber wir haben doch schon Swagger?!?

- CDC beschreibt was der Consumer benötigt
(Definition)



© Pixabay Licence

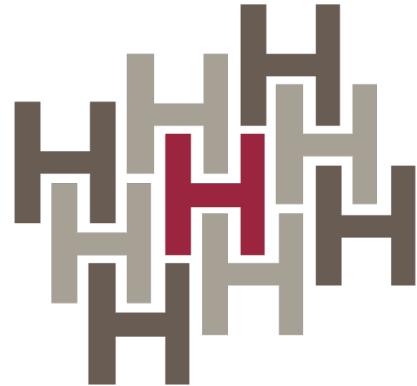
Fazit

- Guter Weg zur Definition minimaler Schnittstellen
 - Der Client bekommt genau was er will (braucht?)
 - Design-by-Contract reloaded
- Entkoppelt die Entwicklung durch automatisierte „Integration ohne Integration“
- Als Mockup für komplexere Integrations-/Akzeptanztests nur bedingt geeignet
- Ideales Werkzeug bei automatisiertem Service Deployment

Links

- PACT
<https://docs.PACT.io>
- PACT JVM
<https://github.com/DiUS/PACT-jvm>
- Mock Server
https://github.com/PACT-foundation/PACT-mock_service
- PACT Broker
https://github.com/pact-foundation/pact_broker
- Beispiele auf github
<https://github.com/wern/cdc-ws-hc-20>

Fragen?



MATHEMA

Vielen Dank!

werner.eberling@mathema.de
(@wer_eb)

PACT

<https://docs.PACT.io>