



# Cucumber.js - Starte den Tag mit BDD!

Holger Grosse-Plankermann

---



# Wer bin ich?

Developer/Consultant/Whatever

Taming the Web since the 2000

Compiled Mozilla for bonus features

Backend vs. Frontend dev



@holgergp



<http://github.com/holgergp>



Holger Grosse-Plankermann

# AGENDA

Um was geht es heute

-  Testen von komplexen SPAs
-  Cucumber.js Technik
-  Bird's eye view
-  Wie fange ich mit Cucumber.js an?

# Montag morgen!

Lasst uns den Tag mit BDD starten!



# Wo stehen wir?

- Wir sind zu Beginn eines Projekts



## Cucumber.js?

# Hände hoch!

Wer von euch kennt Cucumber?

- Cucumberjs?
- BDD?
- JBehave etc.?





Und wer mag es?

# Die Beispielanwendung

Bohnenart	Preis beim Händler in Euro	Marge in Prozent	Preis im Laden in Euro
Äthiopien	10	20	12

- Verwaltung von Kaffee Bohnen und Preisen
- Basiert auf React und Redux
- Editierung eines Wert aktualisiert den Anwendungszustand

# Neue Story

PO 😎: Passt bitte diese Berechnung an.

Barista Pro 1.0			
Bohne	Händlerpreis	Marge	Preis im Laden
Äthiopien	10 €	30%	13€

Wir brauchen da dringend noch ein Feld für einen Rabatt! Geht das?

Entwickler 1 😊: Klaro!

Entwickler 2 😇: Roger!

😎: Na denn mal los!

# An die Tastatur!

😊😊 „Wie fangen wir an?“

😊 „Los hacken! Erstmal ein Eingabefeld basteln!“

😊 „In der Story steht wir geben den Rabatt immer ein!“

😊 „Moment! Gab es da nicht ein Feld im Admin Bereich für?“

😊 „Ich glaube die Berechnung ist mir doch nicht ganz klar.“

😊😊: ⚡ 🔥

😊😊 „Erstmal einen Kaffee!“

# Cucumber.js



😊: „Lass uns mal mit dem einfachsten Fall starten!

*Ich habe schon mal Cucumber.js eingesetzt. Lass uns das hier auch einsetzen..*

**Cucumber.js** ist eine JavaScript Bibliothek, die es mir ermöglicht Tests natürlichsprachlich zu formulieren und auszuführen.

- **Gherkin Syntax**
- **Behaviour Driven Design (BDD)**
- Ausführung im Dev/CI Workflow möglich
- <https://github.com/cucumber/cucumber-js>

## Was heißt BDD?

(nach [Ryan Marsh](#))

Zusammenarbeit im Team



Natürlichsprachliche Beschreibung



Verbunden mit automatisierten Tests



Lebende Dokumentation

# Der erste Cucumber Test

**Funktionalität:** Rabatt für eine Bohnenart berechnen

Als Bohnenverkäufer möchte ich einen Rabatt gewähren können

**Szenario:** Ein neuer Rabatt soll gewährt werden

**Angenommen** es gibt einige Bohnenarten in der Anwendung

**Wenn** der Bohnenverkäufer einen Rabatt von "10" Prozent gewährt

**Dann** ist der Rabatt von "10" Prozent in der Anwendung sichtbar

**Und** ist der neue Verkaufspreis '11.70' Euro

- 😇😊: „Wir starten erstmal hier mit. Ok?“
- PO: ✅ „Aber da fehlt natürlich noch etwas!“

# Welche Probleme gehe ich an

- **Kommunikation zwischen den Entwicklern**

- Verstehe wir die Story richtig?
- Verstehen wir das selbe?
- Wann ist die Story fertig?

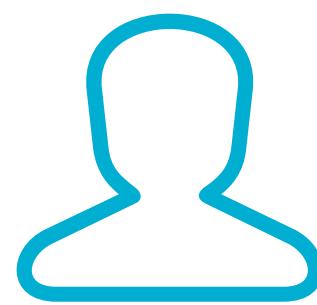
- **Kommunikation zum PO**

- Verstehen die Devs was ich meine?

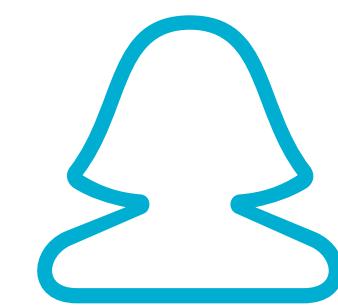
- **Software Test**

- Macht mein neues Feature ein altes nicht kaputt?

# Wo ordnen sich Cucumber Tests ein?



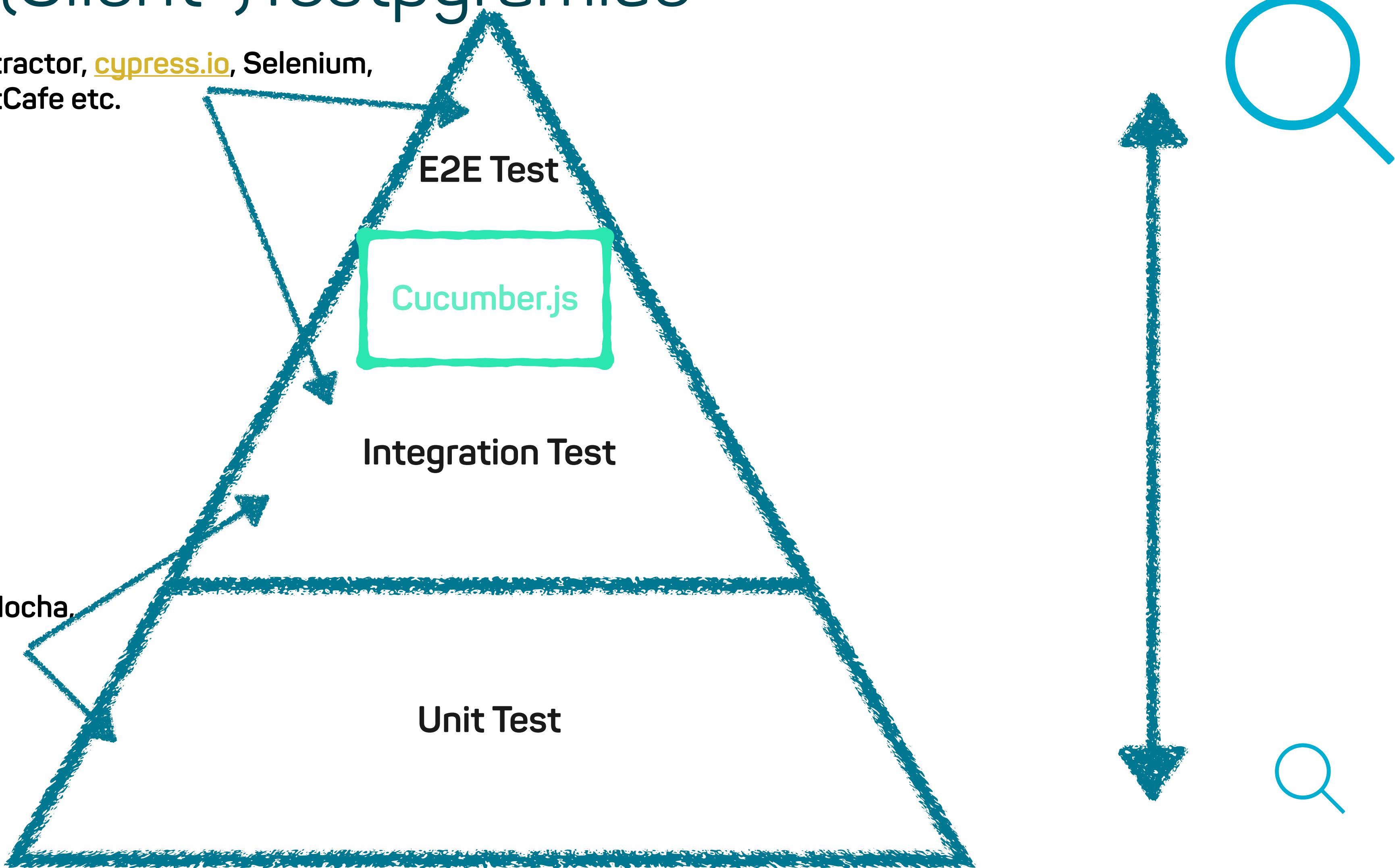
PO



Dev Team

# Die (Client-)Testpyramide

Protractor, [cypress.io](#), Selenium,  
TestCafe etc.



# Ein wenig Technik

Eine kleine Einführung in Cucumber.js



# Der erste Test revisited

**Funktionalität:** Rabatt für eine Bohnenart berechnen

Als Bohnenverkäufer möchte ich einen Rabatt gewähren können

Grobe Beschreibung des Testinhalts.

**Szenario:** Ein neuer Rabatt soll gewährt werden

**Angenommen** es gibt eine Bohnenart in der Anwendung

**Wenn** der Bohnenverkäufer einen Rabatt von "10" Prozent gewährt

**Dann** ist der Rabatt von "10" Prozent in der Anwendung sichtbar

**Und** ist der neue Verkaufspreis '11.70' Euro

Ein einzelnes Testszenario.

Die Syntax nennt sich „Gherkin“

Die einzelnen Testschritte.

Hierfür ist Code notwendig!  
(Step-Definitions)

# Der erste Testlauf

> `yarn test:cucumber`

1) Scenario: Ein neuer Rabatt soll gewährt werden # features/firstTest.feature:6

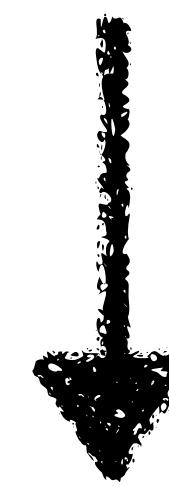
✓ Before # features/support/hooks.js:5

? Angenommen es gibt einige Bohnenarten in der Anwendung  
Undefined. Implement with the following snippet:

```
When('der Bohnenverkäufer einen Rabatt von {string} Prozent gewährt', async function () {
  // Write code here that turns the phrase above into concrete actions
  return 'pending';
});
```

# Die erste Step Definition

```
When('der Bohnenverkäufer einen Rabatt von {string} Prozent gewährt',  
  async function () {  
    // Write code here that turns the phrase above into concrete actions  
    return ,pending';  
  }  
);
```



```
import {Given as Angenommen, Then as Dann, When as Wenn} from 'cucumber';
```

```
Wenn('der Bohnenverkäufer einen Rabatt von {string} Prozent gewährt', function (rabatt) {  
  setInputValue('#rabatt', rabatt);  
});
```



Durch das Umbenennen gleich viel  
leserlicher



Mapping der Eingabeparameter



Keine RegExp nötig! Ist aber möglich.  
Und manchmal sinnvoll

# Das Beispiel erweitert

#language: de  Deutscher Sprachsupport

**Funktionalität:** Bohnenarten in der Anwendung anzeigen

Als Bohnenverkäufer möchte in der Anwendung meine Bohnenarten sehen und bearbeiten können

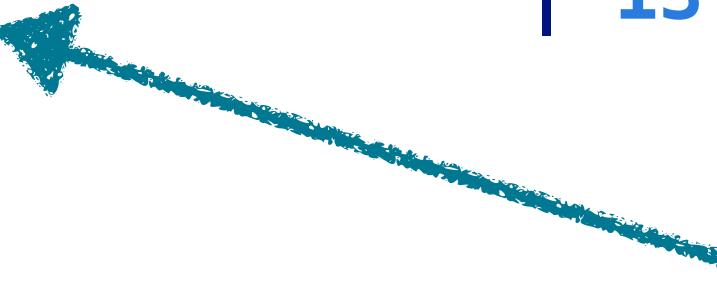
**Grundlage:**  Die Voraussetzung eines Tests braucht eine Step Definition.

**Angenommen** die Anwendung ist geöffnet

**Und** es gibt folgende Bohnenarten in der Anwendung

Id   Bohne   Einkaufspreis in Euro   Marge in Prozent   Verkaufspreis in Euro
1   Äthiopien   10   30   13

**Szenario:** Bohnenarten sind in der Anwendung sichtbar

 Tabellarische Daten machen die Tests lesbarer

# Tabellarische Daten auswerten

```
Angenommen('es gibt folgende Bohnenarten in der Anwendung', function (dataTable) {  
  for (const row of dataTable.hashes()) {  
    const id = row['Id'];  
    const art = row['Bohne'];  
    const ekp = row['Einkaufspreis in Euro'];  
    const vkp = row['Verkaufspreis in Euro'];  
    const marge = row['Marge in Prozent'];  
    this.store.dispatch(updateData({id, art, ekp, vkp, marge}));  
  }  
});
```

Wir bekommen ein dataTable Objekt hereingereicht



Mehrere Möglichkeiten für Zugriff möglich  
Hier: Tabelle mit Header.  
(Pro Zeile ein Objekt mit Headernamen als Key.)

Aktualisierung des Anwendungszustands.



# World

- Globaler Zustand
- Als this verfügbar in allen Step-Definitions

```
import {setWorldConstructor} from 'cucumber';
import {createStore} from 'redux';
import reducers from '../../src/reducer';

function BaristaWorld() {
    this.store = createStore(reducers); ← Applikationszustand (hier: Redux)
}

setWorldConstructor(BaristaWorld);
```

# Szenariogrundriss

Parametrisierbares Szenario

**Szenariogrundriss:** Jede Änderung des Einkaufspreis ist in der Anwendung sichtbar

**Wenn** der Bohnenverkäufer den Einkaufspreis auf "<Einkaufspreis>" Euro setzt

**Dann** sind folgende Bohnenarten in der Anwendung sichtbar

Bohne	Einkaufspreis in Euro	Marge in Prozent	Verkaufspreis in Euro
Äthiopien	<Einkaufspreis>	30	13

**Beispiele:**

Einkaufspreis
39.99
29.99
35.98
36.00
38.99

Hier kommen die Beispiele rein

Hier definieren wir Beispieldaten  
ohne Step-Definitions anpassen zu  
müssen.

# Verzeichnisse

- - ├ cucumber.js       Config-file
  - ├ features       Default für Features
    - | └ bohnen.feature
    - | └ rabatt.feature
    - | └ step-definitions       Step-Definitions
      - | | └ bohnen.js
      - | | └ global.js
      - | | └ rabatt.js
    - | └ support       Helfer und Querschnittliches.
      - | | └ expectations.js
      - | | └ hooks.js       Möglichkeit die Testumgebung global auf- bzw. abzubauen
      - | | └ setupJsDom.js
      - | | └ world.js
  - └ package.json
  - └ src
    - ├ App.js
    - └ reducer.js

Das war jetzt viel  
auf einmal



# Startklar!

Ein neues Feature!



# Ein fehlschlagender Test

- Wir wollen Rabatte einpflegen

Barista Pro 1.0				
Bohne	Händlerpreis	Marge	Rabatt	Preis im Laden
Äthiopien	10 €	30%	0	13€

So in etwa soll das aussehen

# Ein fehlschlagender Test

#language: de

**Funktionalität:** Rabatt für eine Bohnenart berechnen

Als Bohnenverkäufer möchte ich einen Rabatt gewähren können

**Grundlage:**

**Angenommen** die Anwendung ist geöffnet

**Szenario:** Ein neuer Rabatt soll gewährt werden

**Angenommen** es gibt folgende Bohnenarten in der Anwendung

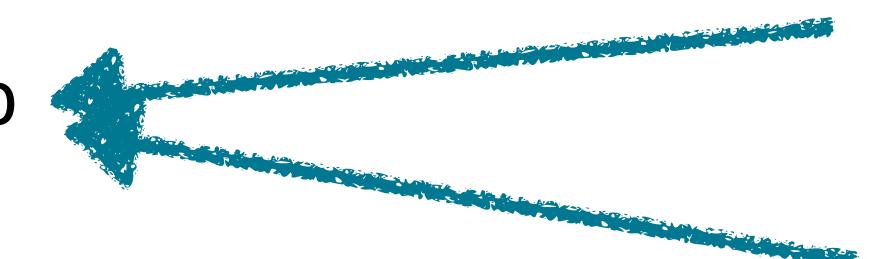
Id   Bohne   Einkaufspreis in Euro   Marge in Prozent   Verkaufspreis in Euro
1   Äthiopien   10   30   13

**Wenn** der Bohnenverkäufer einen Rabatt von "10" Prozent gewährt

**Dann** ist der Rabatt von "10" Prozent in der Anwendung sichtbar

**Und** ist der neue Verkaufspreis '11.70' Euro

„Hmmm. Wahrscheinlich müssen wir hier unterscheiden zwischen dem eingegebenen Preis und dem abgezogenen Rabatt“



„Mal schnell mit dem PO reden.“

# Ein angepasster fehlschlagender Test

#language: de

**Funktionalität:** Rabatt für eine Bohnenart berechnen

Als Bohnenverkäufer möchte ich einen Rabatt gewähren können

**Grundlage:**

**Angenommen** die Anwendung ist geöffnet

**Szenario:** Ein neuer Rabatt soll gewährt werden

**Angenommen** es gibt folgende Bohnenarten in der Anwendung

Id   Bohne   Einkaufspreis in Euro   Marge in Prozent   Verkaufspreis in Euro
1   Äthiopien   10   30   13

**Wenn** der Bohnenverkäufer einen Rabatt von "10" Prozent gewährt

**Dann** ist der Rabatt von "10" Prozent in der Anwendung sichtbar

**Und** ist der Verkaufspreis mit Rabatt '11.70' Euro

**Und** ist der Verkaufspreis '13.00' Euro

# Die Step-Definitions

- Wir haben uns die Step-Definition generiert und diese entwickelt

```
| 1 | Äthiopien | 10 | 30 | 13
Wenn der Bohnenverkäufer einen Rabatt von "10" Prozent gewährt
Dann ist der Rabatt von "10" Prozent in der Anwendung sichtbar

Wenn('der Bohnenverkäufer einen Rabatt von {string} Prozent gewährt', function (rabatt) {
    setInputValue('#rabatt', rabatt);
});

Dann('ist der Rabatt von {string} Prozent in der Anwendung sichtbar', function (rabatt) {
    expect(getInputValue('#rabatt').value).toEqual(rabatt);
});

Dann('ist der Verkaufspreis mit Rabatt {string} Euro', function (vkprabatt) {
    setInputValue('#vkprabatt', vkprabatt);
});
```

# Ein fehlschlagender Test

- Führen wir den Test mal aus:

```
> yarn test:cucumber  
.....F--.
```

Failures:

1) Scenario: Ein neuer Rabatt soll gewährt werden # features/rabatt.feature:8

- ✓ Before # features/support/hooks.js:5
- ✓ Angenommen die Anwendung ist geöffnet # features/step-definitions/global.js:11
- ✓ Angenommen es gibt folgende Bohnenarten in der Anwendung # features/step-definitions/rabatt.js:6


Id	Bohne	Einkaufspreis in Euro	Marge in Prozent	Verkaufspreis in Euro
1	Äthiopien	10	30	13
- ✗ Wenn der Bohnenverkäufer einen Rabatt von "10" Prozent gewährt # features/step-definitions/rabatt.js:17
  - Error: Element for selector #rabatt not found
- Dann ist der Rabatt von "10" Prozent in der Anwendung sichtbar # features/step-definitions/rabatt.js:21
- Und ist der neue Verkaufspreis '11.70' Euro # features/step-definitions/rabatt.js:25
- ✓ After # features/support/hooks.js:9

7 scenarios (1 failed, 6 passed)

28 steps (1 failed, 2 skipped, 25 passed)

Hmmm es gibt noch kein Rabatt Feld.  
Sinnvoller Fehler!

# Das Feature implementieren

- Was müssen wir tun?
- Wir brauchen neue Eingabefelder.

```
<input type="number" id="rabatt" onChange={(event)=> updateRabatt({...bohne, rabatt: event.target.value})}  
      value={bohne.rabatt}/>
```

```
<input type="number" id="vkrabatt" disabled={true} onChange={(event) => updateData({...bohne,  
vkrabatt: event.target.value})} value={bohne.vkrabatt}/>
```

- Und die Berechnung

```
const rabatt = parseFloat(bohne.rabatt || 0.00);  
const vkrabatt = (vkrabatt - (vkrabatt * (rabatt / 100))).toFixed(2);
```

# Ein erfolgreicher Test

yarn test:cucumber

.....  
7 scenarios (7 passed)  
29 steps (29 passed)



Sieht gut aus!

Bohnenart	Preis beim Händler in Euro	Marge in Prozent	Rabatt	Preis im Laden in Euro ohne Rabatt	Preis im Laden
Äthiopien	10,00	30,00	10	13,00	11,70

- Womit machen wir weiter:
  - Admin Rabatt?
  - Soll der Preis nach Rabatt editiert werden können?
  - Etc. PP

# Wie greife ich auf die UI zu?

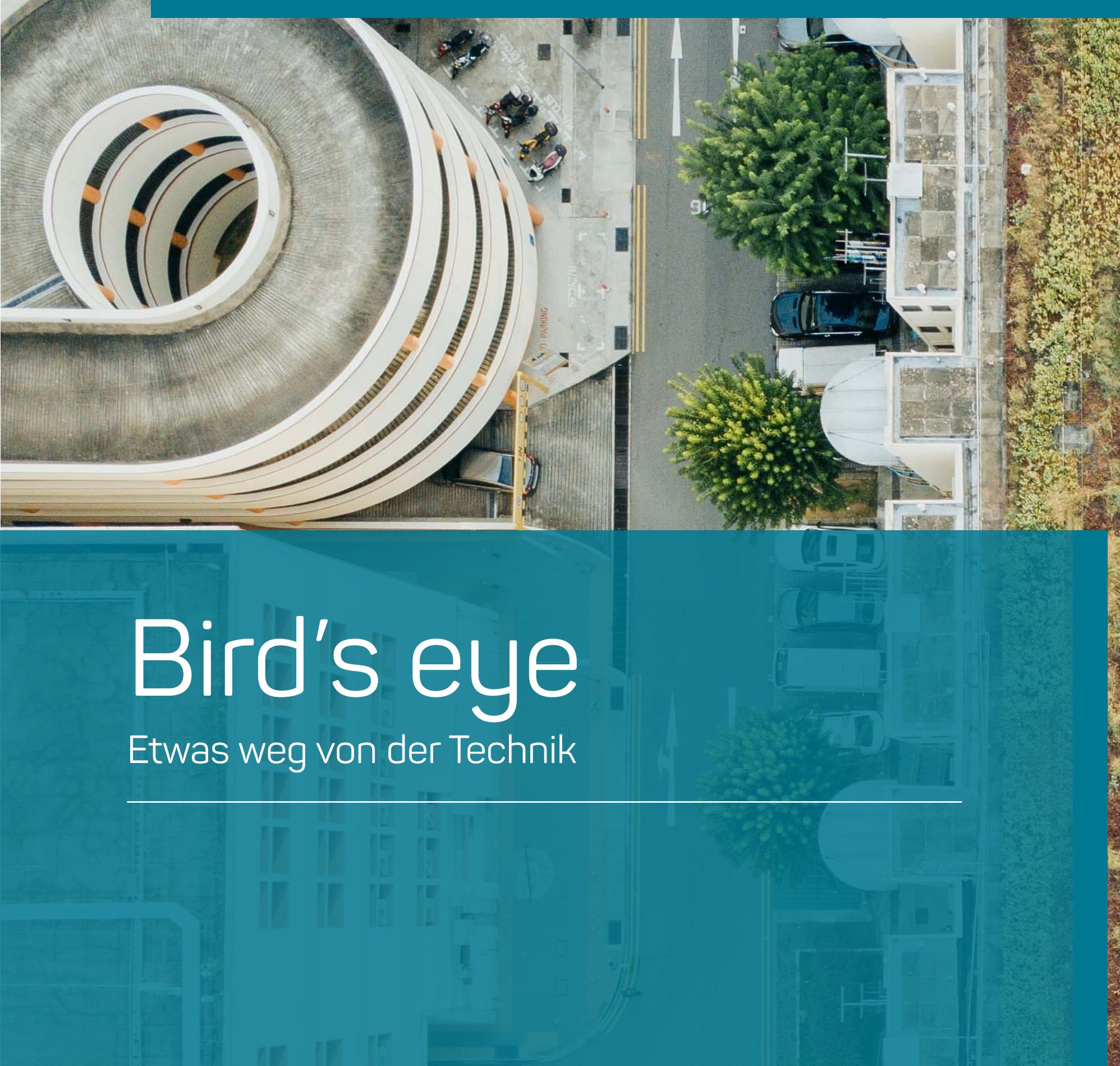
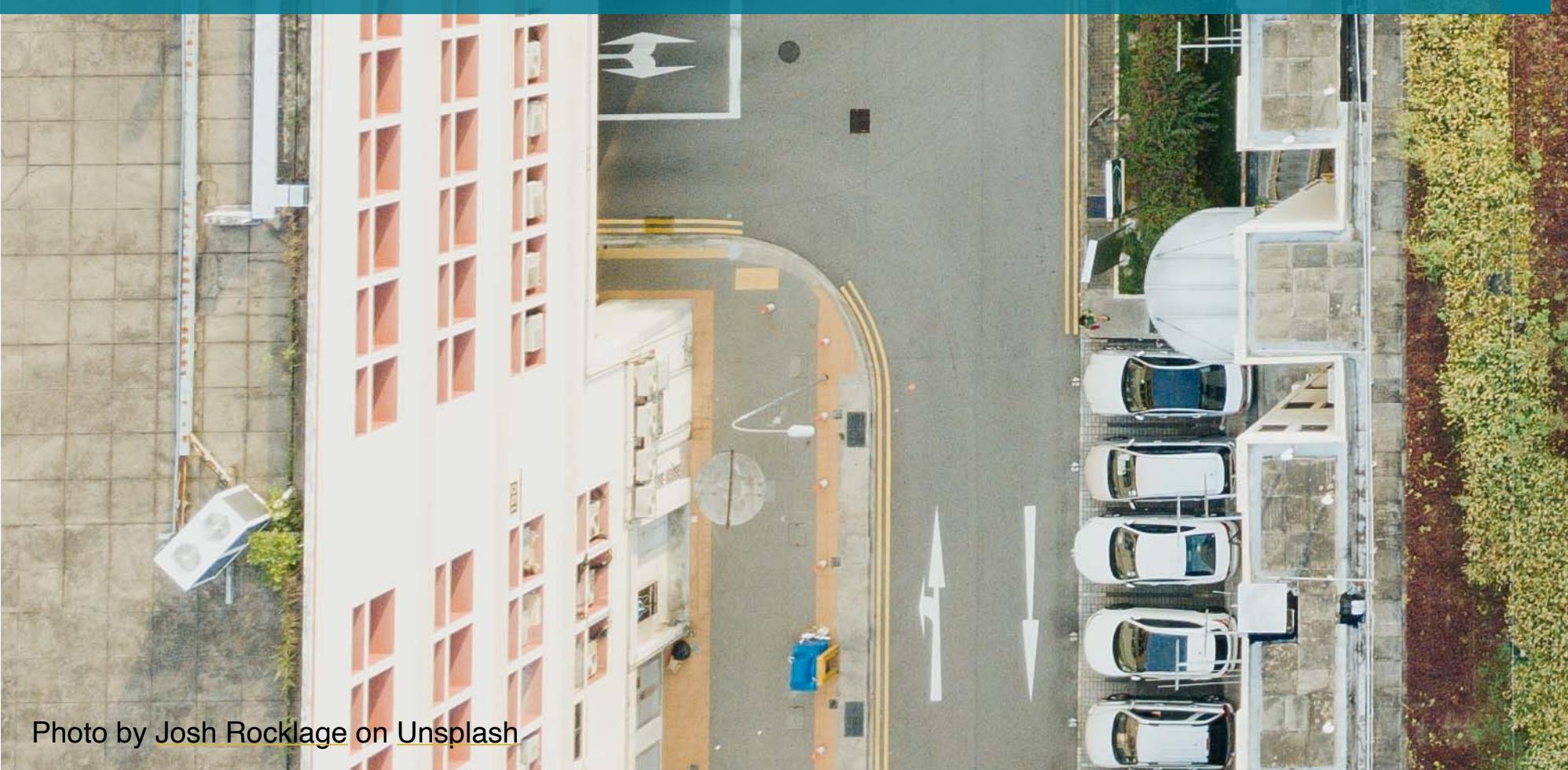
- In unserem Fall reichte uns die Möglichkeiten die Enzyme, React-Test-Utils und die DOM-API lieferten.

```
export function setInputValue(selector, value) {  
  const input = document.querySelector(selector)  
  ReactTestUtils.Simulate.focus(input);  
  input.value = value;  
  ReactTestUtils.Simulate.change(input, {target: {value}});  
  ReactTestUtils.Simulate.blur(input);  
}
```

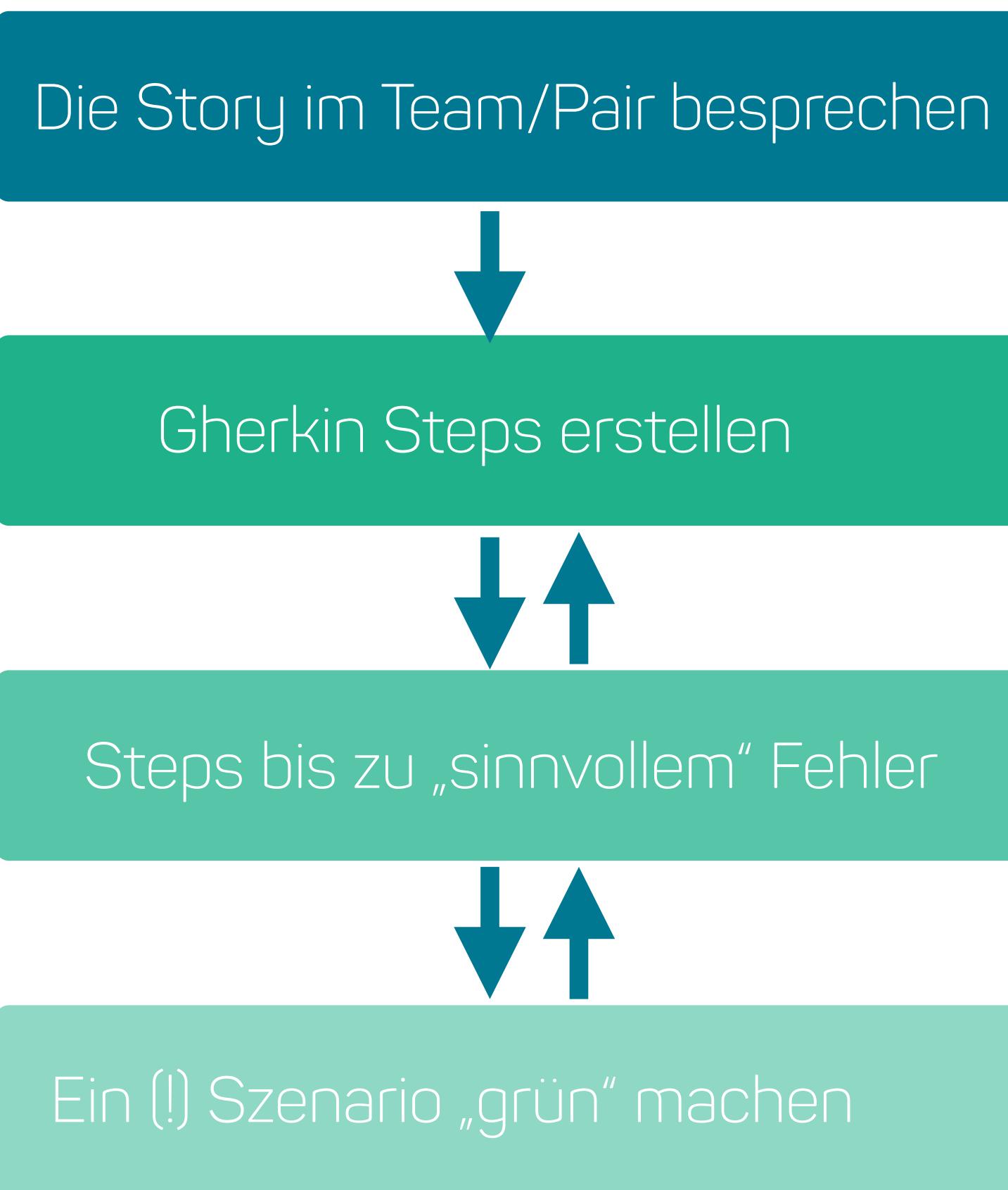
- Aber vielleicht muss man im konkreten Fall ja gar nicht auf die UI zugreifen.

# Bird's eye

Etwas weg von der Technik



# Wie sieht der Arbeitsfluss aus?



Dieses Vorgehen ist auch als

**Acceptance Test Driven Development (ATDD)** bekannt

**Definition der Steps kann iterativ erfolgen**

**UI Element nicht vorhanden**

**Erwartete Berechnung falsch**

**Ein Szenario fertig entwickeln**

**TDD hat uns auch hier geholfen**

# Testlaufzeit

- Die Tests die wir formulieren, haben einen recht großen Scope (den ganzen Client)
- Tests auf diesem Scope haben eine längere Laufzeit als Unit Tests
  - vgl. Testpyramide
  - Bei ca. 600 Tests > 20 Minuten
- Dafür hatten wir zwei Strategien um damit umzugehen.
  - **Teststruktur**
  - **Parallelisieren der Tests**

# Teststruktur



## Vorbereitung des Tests (Arrange/Given)

- Nicht über die UI
- Erspart lange Klickwege
- Schneller
- Anders als bei klassischen UI Tests



## Ausführung (Act/When)

- Fall zu Fall Entscheidung
- Bei wirklicher Nutzeraktion über UI
- Bei externer Aktion Event generieren (z.B. Redux Saga)



## Überprüfung (Assert/Then)

- Oft in der UI
- Abweichen falls nicht sichtbar

# Parallelisieren

- Cucumber.js führt Tests im Default parallel aus
  - Das macht es auf dem eigenen Rechner schneller
- Im CI haben wir die Testausführung aufgeteilt (Single CPU System)
  - Tests A-B
  - Tests C-E
  - ....
- Im Angular Beispiel auf Jest-Mittel zurückgreifen

# E2E or not E2E

- In unserem Fall hatten wir fast alle Logik auf dem Client und nur wenig Serverzugriff
  - Start ohne Server
  - Mocken der Serverendpoints
- Unsere Tests waren dadurch stabil
  - Es gab überraschend wenig Flakyness
  - Ein „Moving Part“ weniger
- Wir haben für ausgewählte Tests einen Testserver verwendet
  - Diese waren deutlich mehr flaky
- **Wir waren mit diesem Trade-Off sehr zufrieden und können es weiterempfehlen!**

# E2E

Falls doch E2E erwünscht ist (...weil es der Kontext erfordert)

- Verzichtet auf Selenium.
- Nutzt Cucumber Plugins für [cypress.io](#) oder TestCafe
- Aber E2E macht eure Tests um eine Größenordnung langsamer und wackeliger
- Achtet darauf, dass die Laufzeit und die Stabilität nicht die **Akzeptanz** dieser Tests untergräbt

# Wie fange ich an?

Kann mir cucumber.js helfen?

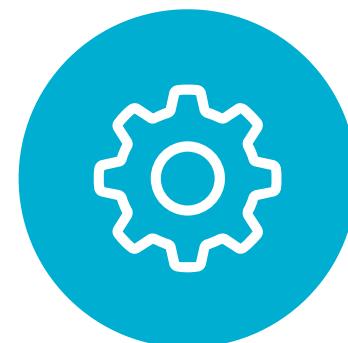


# Bringt Cucumber.js für mich einen Mehrwert?

- Es bringt aus meiner Sicht meistens einen Mehrwert..
  - Insbesondere bei umfangreichen Clients
  - Noch mehr bei komplexer Fachlichkeit (aber wann ist die Fachlichkeit mal einfach?)
- Für mich wichtige Punkte hier sind:



Kommunikation



Bessere Testautomatisierung

# Cucumber.js und Kommunikation

- Intensivere Auseinandersetzung mit einer Story auf fachlicher Ebene



## Im Pair

Gemeinsames Verständnis im Pair, was man eigentlich machen möchte.  
Dies ist für mich Gold wert!



## Solo

Ebenso wertvoll, um sich selber diese Klarheit zu schaffen



## Mit dem PO/Fachabteilung

Die Ergebnisse als Klartext zu formulieren hilft bei fachlichen Diskussionen. Der Problemkontext wird viel klarer.

# Cucumber.js und Testautomatisierung

- Neben den kommunikativen Vorteilen ist der Aufbau einer Akzeptanztest-Suite ein Gewinn



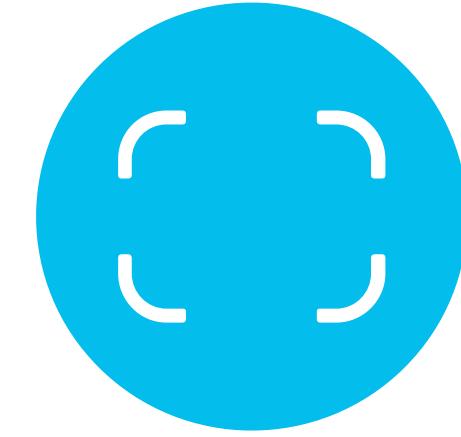
## Vertrauen in die Tests

Durch die Stabilität haben wir ein großes Vertrauen in die Aussagekraft der Testergebnisse entwickelt



## Turnaround

Durch dieses Vertrauen brauchten wir viel weniger auf die laufende Anwendung zurückgreifen



## Klarerer Kontext

Regressionen können aus meiner Sicht so viel besser gefangen und gelöst werden. So wird der Kontext klarer, als wenn man die Test rein technisch formuliert.

# Wie ging das bei uns los?

😊 „Wollen wir das nicht mal ausprobieren, das könnte folgende **Probleme** lösen!?”

🤔🤔🤔 „Ist das nicht total viel **Overhead**?“

📦 „Ich war nicht immer **PO**. Hab das auch schon eingesetzt!“



# Welche Probleme wollten wir mit Cucumber.js adressieren?

- Features wurden nicht fertig weil Dinge hineininterpretiert wurden.
- Es wurden schlicht Sachen vergessen zu implementieren.
- Ein fachlicherer Blick war von Nöten
- Die Stabilität des Clients als Ganzes sollte gewährleistet werden.

**Goldplating**

*„Das muss man doch generalisieren!“*

**Explizit vs. implizit**

**Regressionen vermeiden**

**Der Client war wirklich umfangreich**

# „Ist das nicht großer Overhead?“

- Anfangs ist wie bei jeder Neuerung etwas invest nötig, klar
- Relativ schnell hat sich bei uns daraus aber ein Automatismus entwickelt, der einfach dazu gehörte
- Der Moment an dem wir zum ersten Mal ein Feature nur aus vorhandenen Steps beschreiben konnten war sehr schön.
- Bei uns hat sich recht schnell ein Vertrauen in die Cucumber-Tests eingestellt.

**Hat sich das für uns gelohnt?**

*Zitat Holger als wir mit Cucumber.js losgelegt haben*

**Neue Technik**

**Teststrategie**

**Steps implementieren**

**Wir waren uns im Team dann sehr klar wann eine Story fertig ist.**

**Wir mussten den Client viel seltener von Hand durchklicken,**

***Klares Ja!***

# Reichen da nicht unsere anderen Tests?

- Welche anderen Tests gibt es denn bereits?
- Welche Probleme möchte man lösen?

## Unit Tests

- Turnaround schneller
- Kleinerer Fokus

## Integration Tests/E2E-Tests

- Ist schon eine ausreichende Basis da?
- Brauchen wir nur Tests zum Fangen von Regressionen?
- Manchmal sind technischere Tests greifbarer

- Es entsteht durchaus Aufwand und der muss sich lohnen und zum Problem passen.

# Wie sag ich's dem Team oder dem PO?



## Entwickleraufgabe

Ähnlich wie Unit Tests  
Initialer Mehraufwand



## Problem greifbar

Person/Team sollte das  
zu lösende Problem  
griffig beschreiben  
können



## Guerilla Taktik

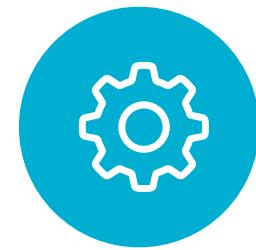
Ein Pair/Entwickler startet  
Einfache Aufgabe  
Größere Kreise ziehen  
Dann erfolgt die Akzeptanz Richtung  
PO/Team schneller

# Muss es denn all-in sein? Nein!



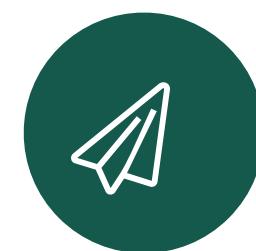
## Auch für kleinere Slices nützlich

z.B. ein kleiner Teil der Anwendung stellt sich als herausfordernd dar



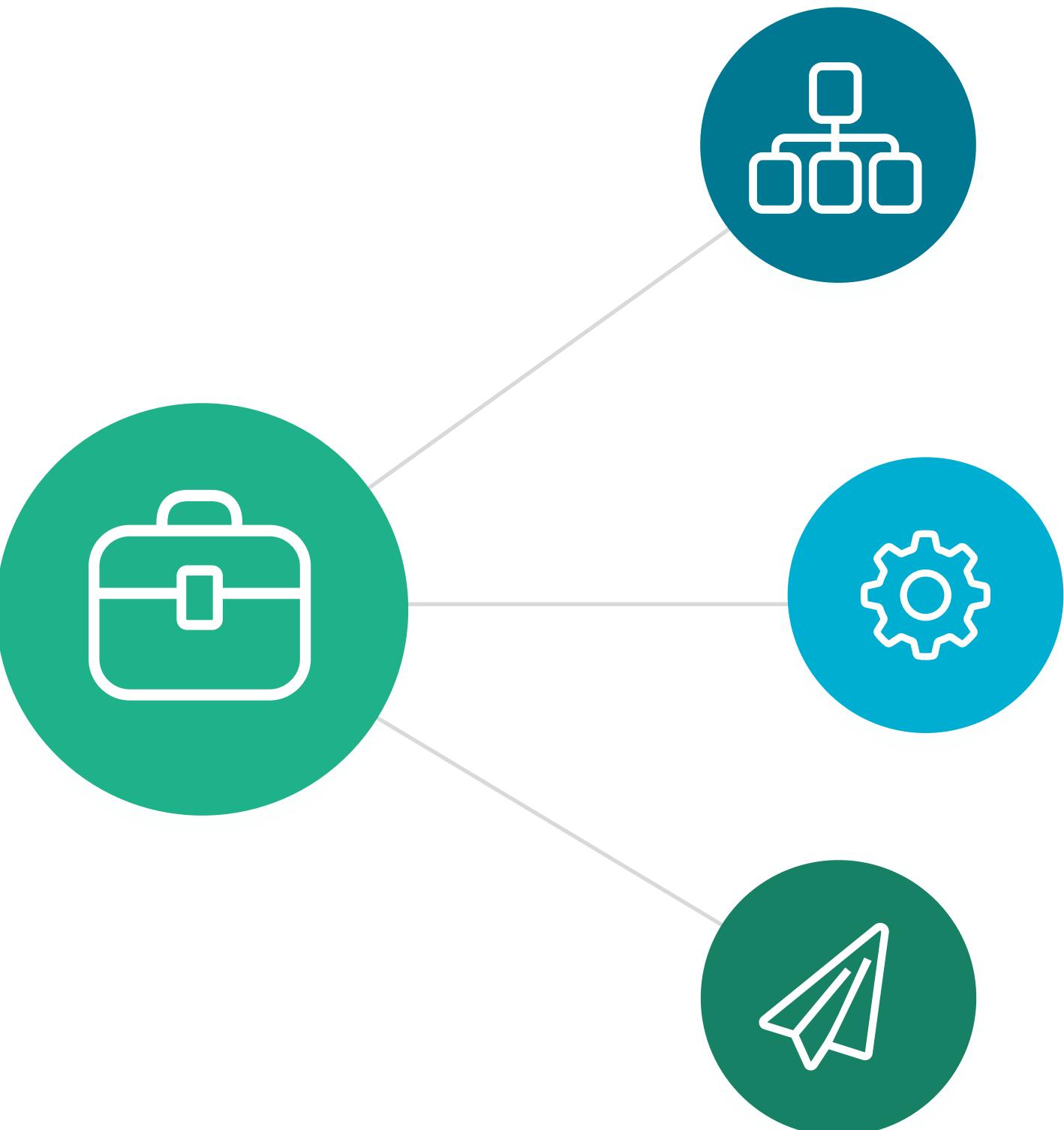
## Man kann mit technischen (Unit-) Tests starten

Wenn das funktioniert, super!



## BDD-light Ansatz

Unterstützt Entwicklung und macht für Domänen Experten und Techniker Aufgabe greifbarer



# Fazit

---

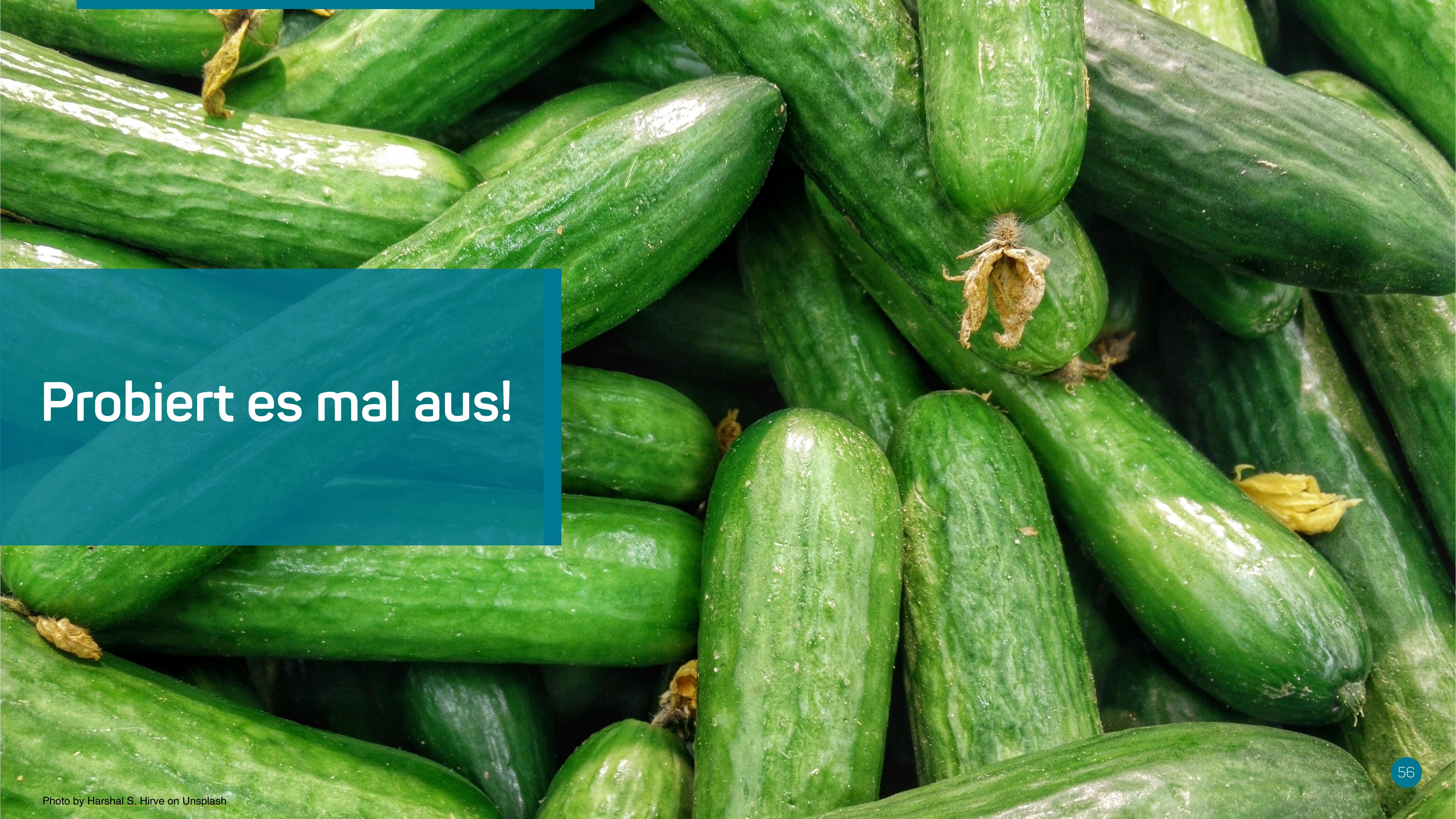


# Fazit

- In meinem Kontext hat Cucumber.js super funktioniert
- Andere Herangehensweise an eine Story und verbesserte Kommunikation
- Besserer Turnaround
- Sehr gutes Sicherheitsnetz um Regressionen in der Anwendung zu fangen

# Was gibt es noch

- jest-cucumber
  - <https://www.npmjs.com/package/jest-cucumber>
- Cypress Cucumber Preprocessor
  - <https://github.com/TheBrainFamily/cypress-cucumber-preprocessor>
- Testcafe Cucumber
  - <https://github.com/rquellh/testcafe-cucumber>
- Beispiel Sourcen
  - <https://github.com/holgergp/cucumberjsTalk/tree/master/example-app>
- Beispiel für Cucumber mit Cypress
  - <https://github.com/holgergp/cypressioCucumberExample>



Probier es mal aus!



A close-up photograph of a cup of coffee with a latte art heart on a dark wooden surface. The cup has a green handle. In the top left corner, there is a teal-colored rectangular overlay containing the text "Danke fürs Zuhören!" in white.

Danke fürs Zuhören!

# Stay connected

Danke fürs Zuhören!

Our mission – to promote agile development, innovation and technology – extends through everything we do.



## Address

codecentric AG  
Hochstraße 11  
42697 Solingen



## Contact Info

E-Mail: [info@codecentric.de](mailto:info@codecentric.de)  
[www.codecentric.de](http://www.codecentric.de)



## Telephone

Telefon: +49 (0) 212. 23 36 28 0  
Telefax: +49 (0) 212.23 36 28 79



# Backup

Ein neues Feature!

---

# Mehr Config

cat /example-app/cucumber.js

```
const babel = '--require-module @babel/register'; ← Hier verbinden wir Cucumber.js mit Babel.  
const asyncAwaitSnippets = ← Hier mit können wir die Snippet Syntax kontrollieren  
  '--format-options \'' + JSON.stringify({'snippetInterface': 'async-await'}) + '\';  
const now = '--tags @now'; ← In diesem Test werden nur Tests ausgeführt, die  
  mit dem Tag @now markiert sind  
module.exports = {  
  'default': [babel, asyncAwaitSnippets].join(' '),  
  'now': [babel, now, asyncAwaitSnippets].join(' ')  
};
```

@now

**Szenario:** Ein neuer Rabatt soll gewährt werden

**Angenommen** es gibt einige Bohnenarten in der Anwendung

# Unterschiede zu Cucumber.java

- Cucumber.java hat in meinem Kontext keine sonderlich große Beliebtheit, wo sind die Unterschiede:
- Regular Expressions in Step-Definitions sind optional
  - Das hat mir den Einstieg sehr erleichtert
  - Mittlerweile ist dies auch bei Cucumber.java so
- Die World als Verbindungsglied zwischen den Tests hat ein sehr einfach verständliches Konzept
  - Und kommt Out-of-the-Box
  - Leichter Einstieg
- Leichte Wiederverwendbarkeit der Steps
  - Wir sind nicht mit der Einschränkung gestartet, dass wir pro Test die Steps neu implementieren

# Wie binde ich cucumber.js ein

```
{  
  "dependencies": {  
    . . . React/Redux  
  },  
  "devDependencies": {  
    . . .  
    "expect": "^24.9.0",  
    "cucumber": "^6.0.5", ← Cucumber.js Bibliothek  
    "gherkin-lint": "^3.10.0" ← Ein Linter für die Gherkin Syntax  
  },  
  "scripts": {  
    "gherkin": "node_modules/.bin/gherkin-lint",  
    "test:cucumber": "node_modules/.bin/cucumber-js",  
    "test:cucumber:now": "npm run test:cucumber -- -p now", ← Über Tags können wir nur bestimmte Tests ausführen  
  }  
}  
  @now  
  Szenario: Bohnenarten sind in der Anwendung sichtbar
```

# Hooks

- Bietet die Möglichkeit die Testumgebung global aufzubauen und/oder aufzuräumen

```
import {After, Before} from 'cucumber';
import setUpJsDom from './setupJsDom';
```

```
Before(function () {
  this.root = setUpJsDom();
});
```

Vor jedem Testlauf wird hier im Beispiel eine Enzyme Umgebung aufgebaut...

```
After(function () {
  _unmountUI(this);
});
```

... und nach dem Test wieder heruntergefahren