

# Cypress

Die Kunst der simplen E2E Tests

# Agenda

a]

- Grundlagen Testing
- Cypress Intro
- Übungen

# HANNAH



# MILENA



# Kurze Fragerunde

Go to

**www.menti.com**

Enter the code

**5833 8826**



Or use QR code

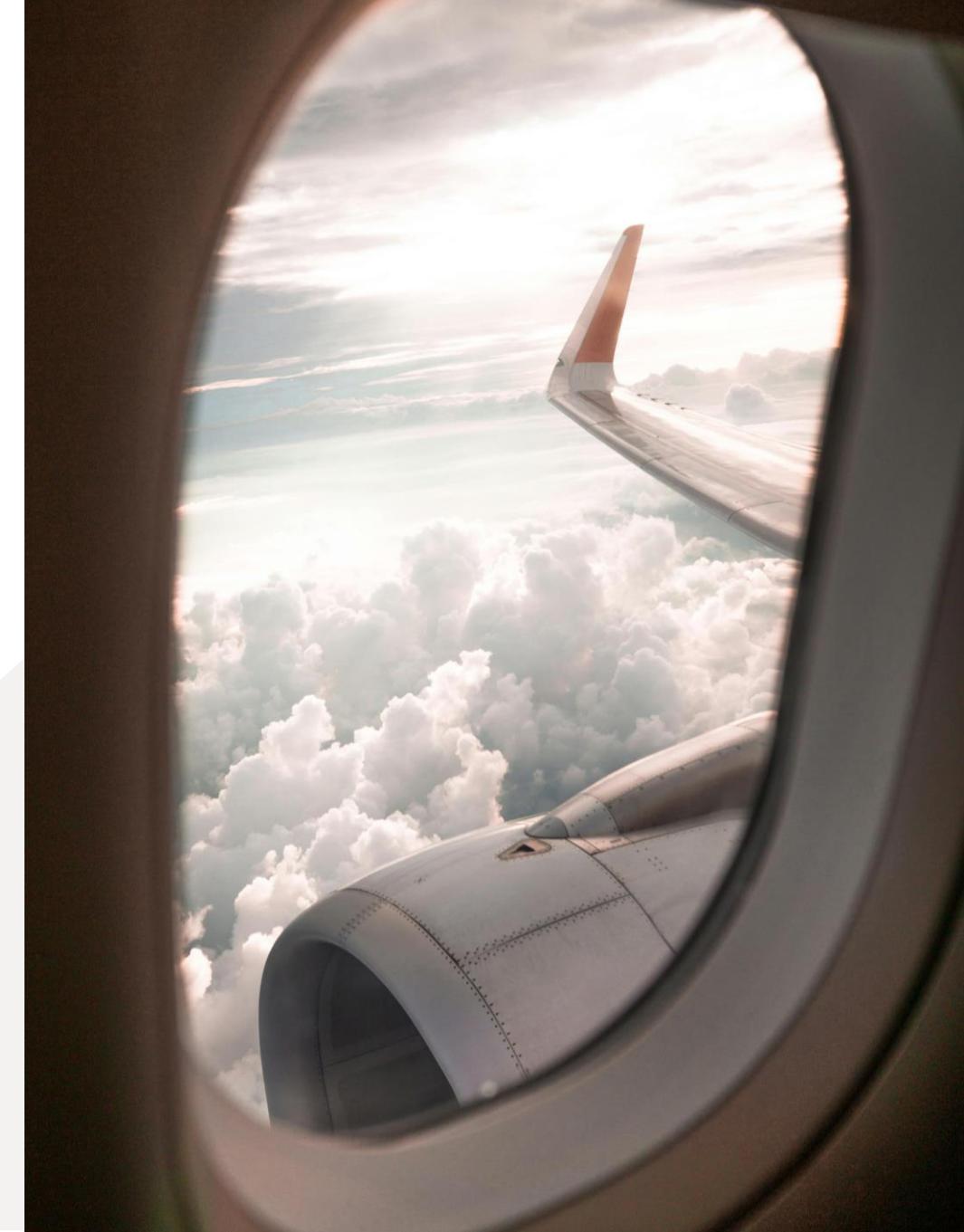
# Testen

## – warum testen wir überhaupt?



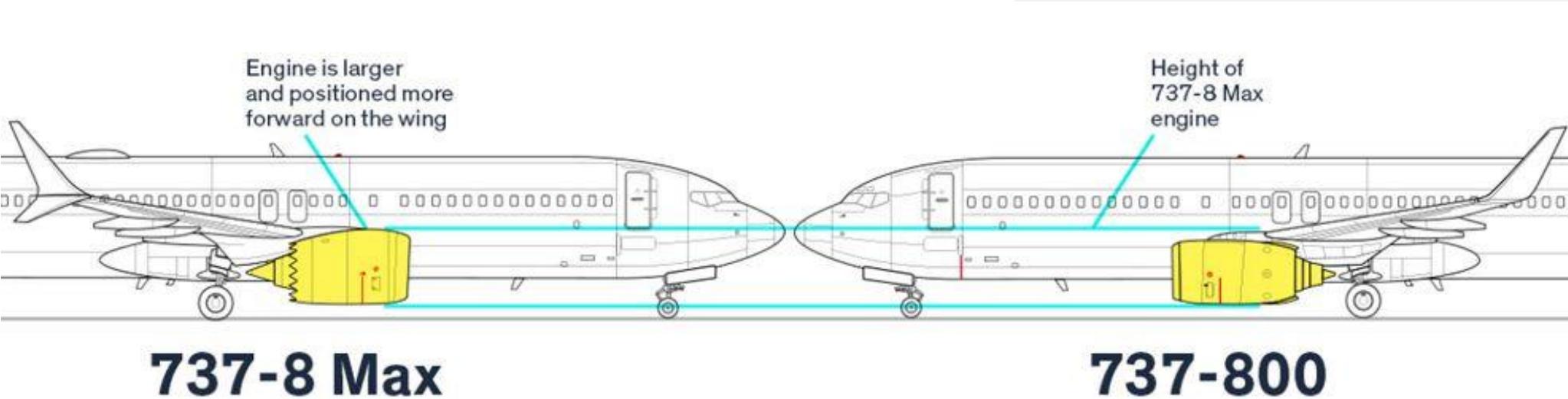
# Der Boeing-Fall 737 Max

- Ziel: Die "Seat-Mile-Kosten" so gering wie möglich halten
- Ein größerer Motor verbraucht weniger Kraftstoff pro Leistungseinheit
- Lösung: ein größerer Motor bei gleicher Flugzeuggröße



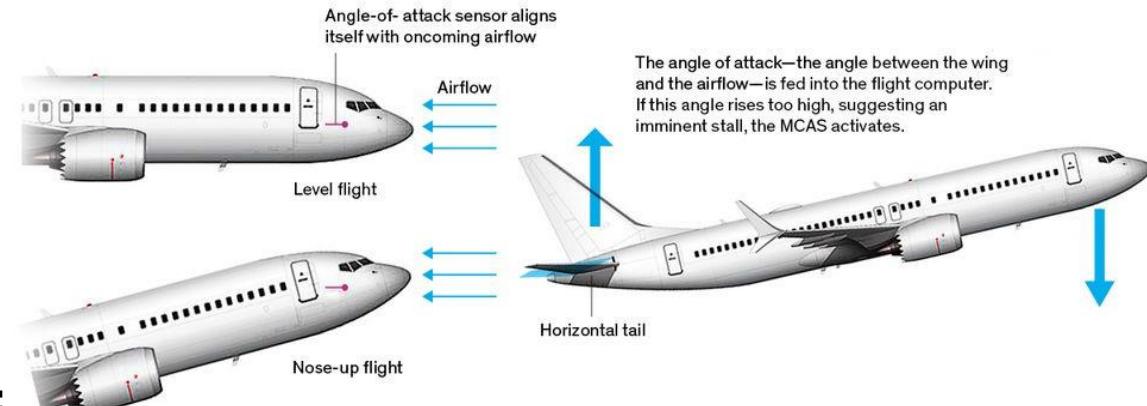
# Die Folgen

- Schwerpunkt des Motors verschiebt sich
- Flugzeug neigt dazu mit der Nase nach oben zu zeigen



# Software zum Ausgleich

- Um die unpassende Hardware auszugleichen, setze Boeing auf "Maneuvering Characteristics Augmentation System" (MCAS)
- MCAS drück die Nase des Flugzeugs runter, wenn es denkt der Angle-of-attack ist zu groß
- Flugzeuge werden von Millionen Zeilen Code gesteuert!



# Testing Failure

- Software-Entwickler sind nicht vertraut mit der Luftfahrt und physikalischen Gesetzen
- Piloten waren nicht informiert und konnten Automatismus nur schwer übergehen
- Wie konnte der Fehler beim Testen und unter simulierten Testbedingungen nie auftreten?



# Folgen: Testen auf Prod

- 346 Tote nach Abstürzen diverser 737 Max Flugzeuge
- Finanzieller Schaden von > \$18,7 Milliarden für den Hersteller
- Einbruch der Umsätze
- Ist schlechte Hardware in Kombination mit schlechter Software und Auslassen teurer Tests nicht doch insgesamt günstiger?

# Warum Testen wir also?

- Tests kosten Zeit und Ressourcen
- Warum glaubt ihr, sollten wir dennoch Testen? (Egal, welche Form von Tests)



Testing leads to failure,  
and failure leads to understanding.

Burt Rutan (Luft- und Raumfahrt ingenieur)



# 5 Gründe zu testen



1.

Ihr werdet dankbar für  
jeden Hotfix sein, der nicht  
gefixt werden muss.



2.

Eure Kollegen werden es  
euch danken, wenn Sie  
wenn sie neue  
Funktionen und  
Features hinzufügen.



3.

Es führt zu besser  
strukturiertem und  
wartbarem Code.



4.

Grenzfällen werden  
beachtet und  
abgetestet.



5.

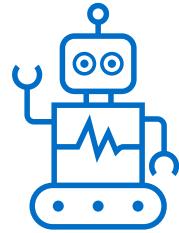
Ihr dokumentiert  
automatisch  
euren Code.



# Testen was ist das?



# Manuell oder automatisiert



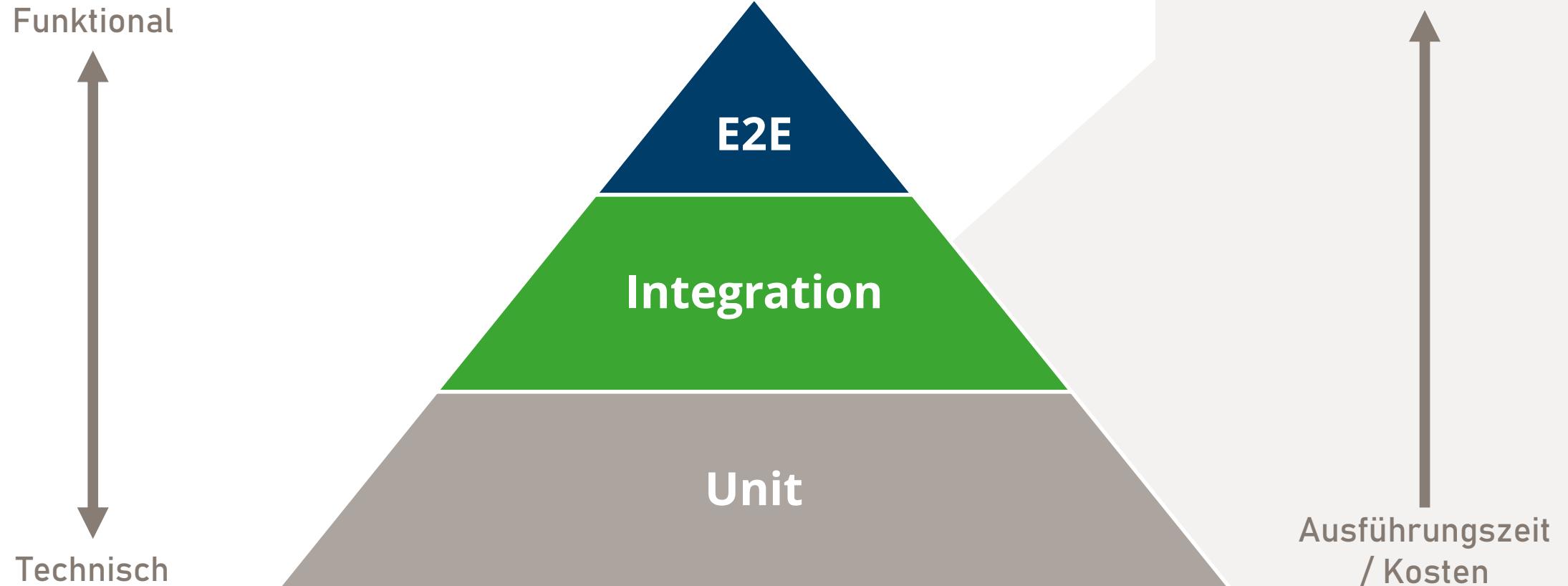
## Automatisiertes Testen

- Schnelligkeit
- Wiederholbarkeit (Regression)
- Integration in Build Pipeline
- Hohe Testabdeckung
- Initialaufwand

## Manuelles Testen

- Stetig hoher Aufwand
- Hohe Fehleranfälligkeit

# Automated Testing

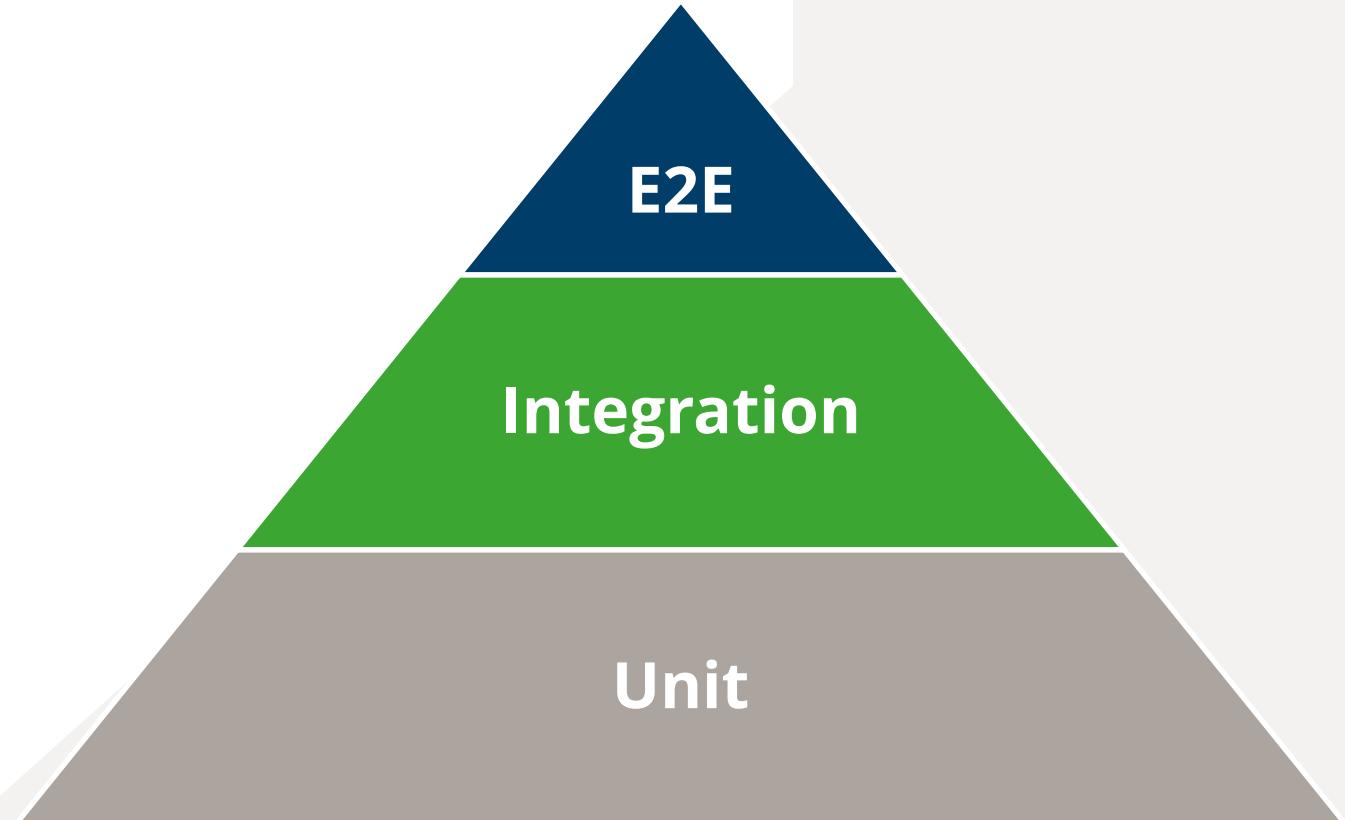


# Automated Testing

## Unterschiede liegen in

- Implementierungsaufwand
- Ausführungsaufwand
- Fehlerauswertung
- Realitätsnähe
- Code Coverage
- Testfallabdeckung
- Fachlich vs. technisch

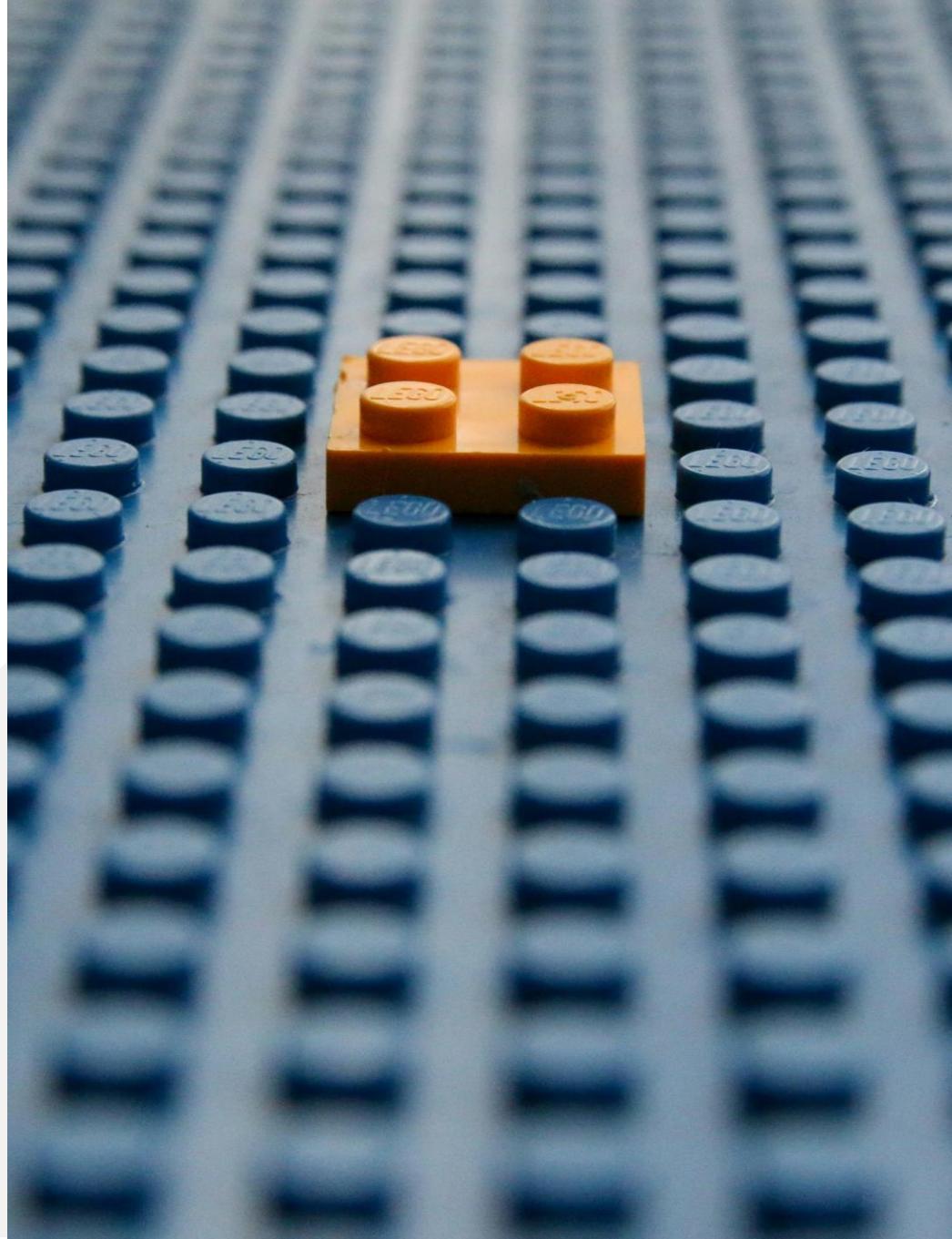
=> Balance ist  
ausschlaggebend



# Was ist eine "Unit"?

Der kleinste Baustein einer Applikation

- Funktion
- Klasse



# Grundannahme

Wenn alle Units  
funktionieren, dann funktioniert  
auch die Applikation.

=> Units sind einfach  
automatisiert zu testen



# Was heißt Integration?

- Verknüpfung von voneinander unabhängigen Komponenten
- Bspw. Verknüpfung von Daten, Anwendungen, APIs und Geräten



# Integration testen

Funktionieren die einzelnen Komponenten auch im Zusammenspiel miteinander?



# Fachlichkeit testen

- Oft wollen wir fachliche Geschäftsabläufe ganzheitlich verifizieren
- Ziel: Aufdeckung von Fehlern aus der Benutzersicht



# Ein System als Ganzes testen

- Vom Web Browser bis zum Backend
- Integrationen mit externen APIs und Services testen
- Testen, ob ein System in verschiedenen Umgebungen funktioniert (z.B. Browser)

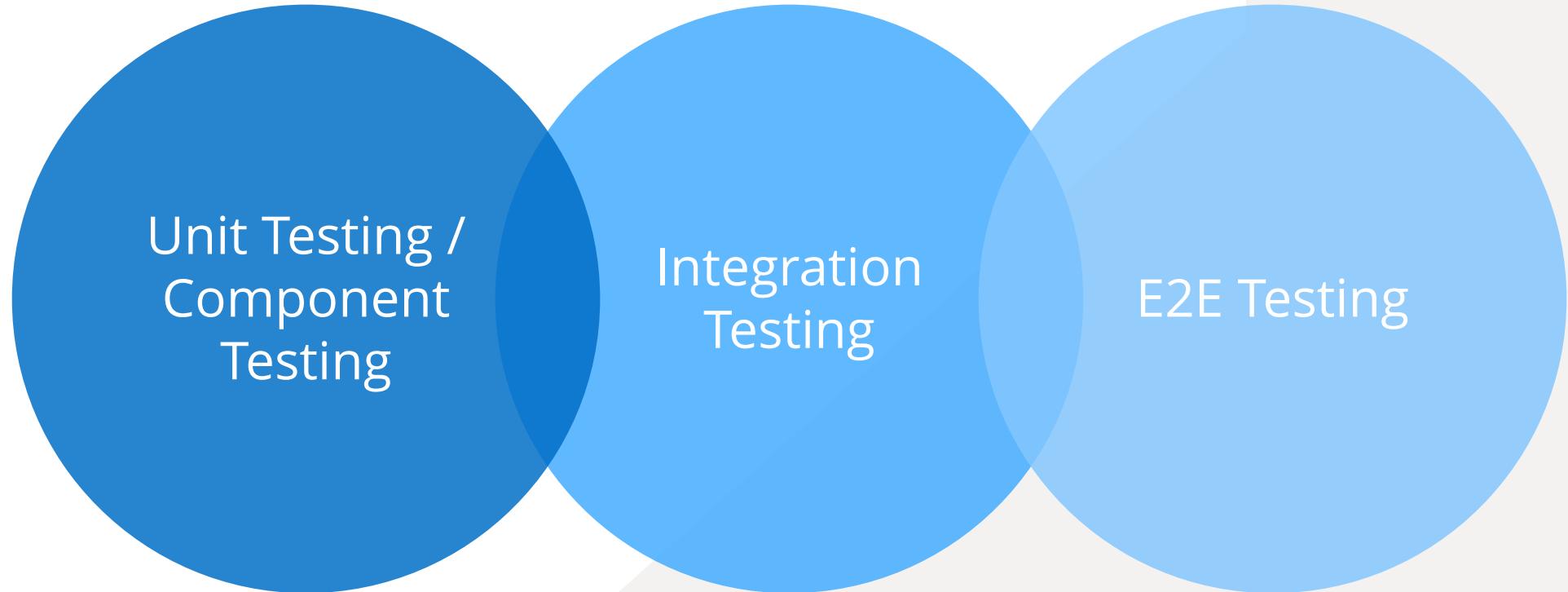


# End-to-end-Tests

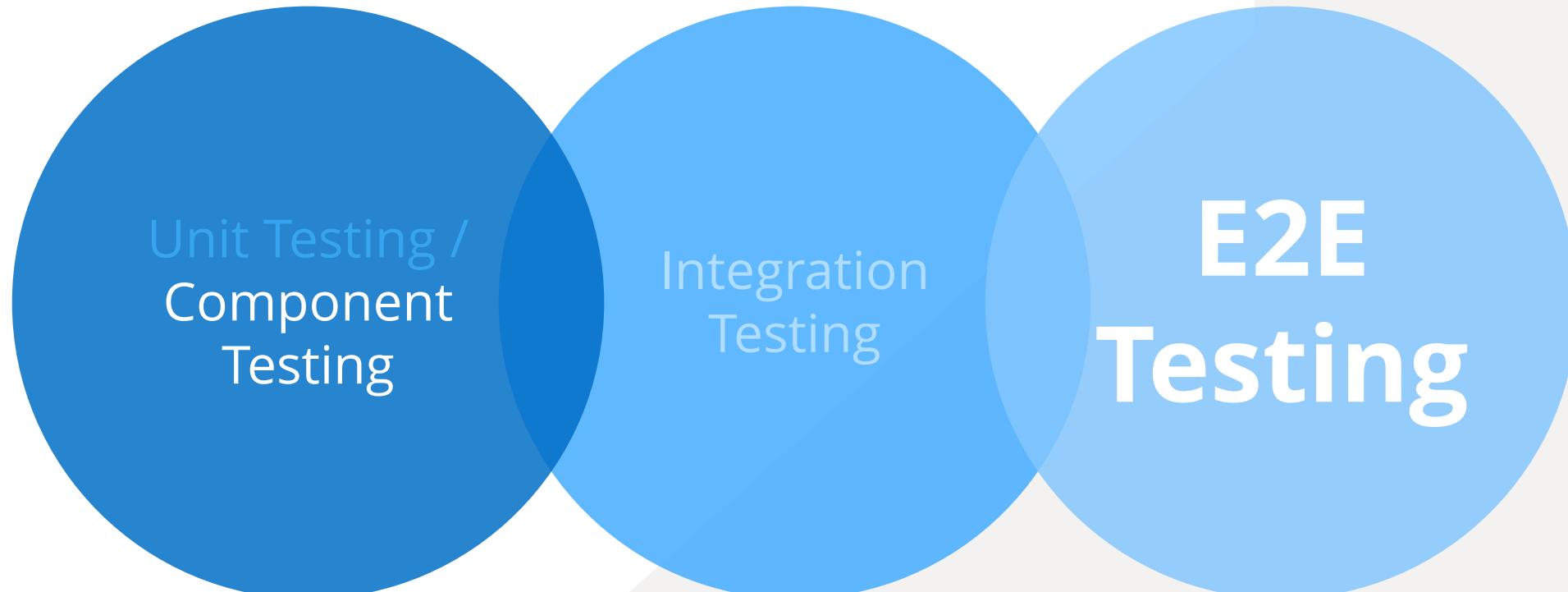
- Simulieren das echte Nutzererlebnis
- Werden oft von speziellen QA-Teams geschrieben
- Sind teuer und haben lange Ausführungszeiten => müssen sich lohnen



# Keine harten Grenzen



# Und was macht Cypress?



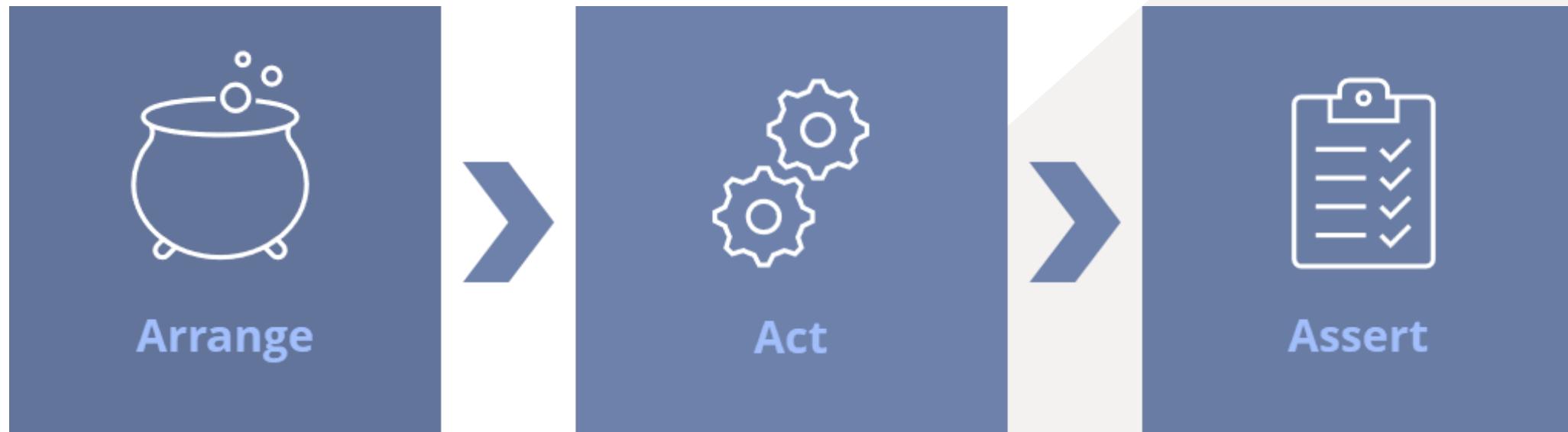
# Testen

## – mal so ganz generell

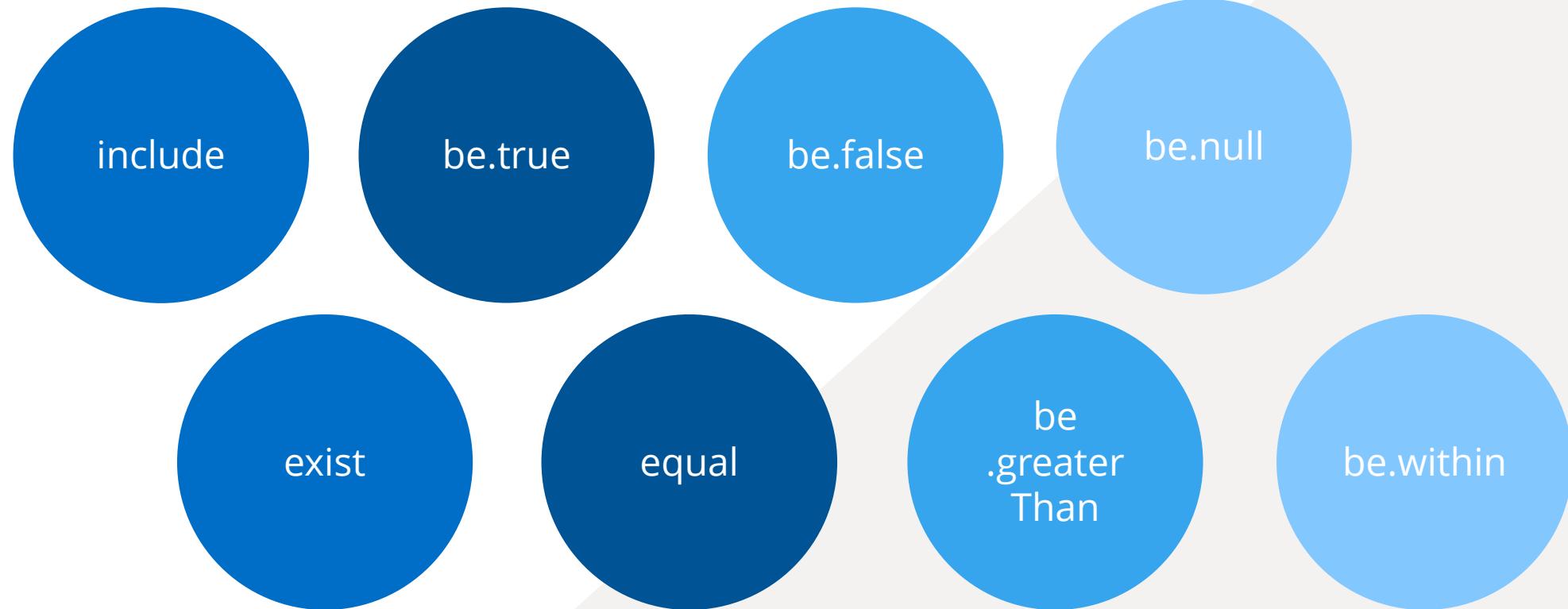


# Das Triple-A-Prinzip

Ein guter Test ist nach dem Prinzip Arrange-Act-Assert aufgebaut.



# Erwartungen formulieren – mit Chai



# Hooks

## Problem

Ein Testfile ist auch nur irgendein Javascript File.

Um doppelten Code zu vermeiden, könnte man eine globale Variable wiederverwenden.

In der Dateiausführung kämpfen spätere Tests mit Variablenveränderungen vorher ausgeführter Test.

## Lösung

Test Lifecycle Hooks

# Hooks



# Tests strukturieren

Testsuite: `describe()` oder `context()`

Test: `it()` oder `specify()`

Nur einen Test laufen lassen: `it.only()`

Einen Test überspringen: `it.skip()`

Join at [menti.com](https://menti.com) | use code 5833 8826

Mentimeter

# Quiz

Go to

**www.menti.com**

Enter the code

**5833 8826**



Or use QR code

a]



The logo features the word "cypress" in a bold, dark navy blue sans-serif font. To the left of the text is a circular graphic element consisting of two concentric arcs. The inner arc is a darker shade of teal, and the outer arc is a lighter shade of mint green. The word "cypress" is partially overlapping the right side of the circular graphic.

cypress

# Developer Experience

Die Cypress Philosophie



# Batteries included - Das Cypress Setup

- Testing Framework: Mocha
- Assertion Library : Chai
- Launcher/ Test Runner
- Reporter
- Browser: Electron
- Proxy (für Request Stubbing)



# Die Cypress Desktop App

- Selector Playground vereinfacht das Schreiben der Tests
- Visuelle aufbereitetes Feedback
- Timetravel
- Übersichtliches Debugging



# Sehr einfaches Mocking

→ Sehr nützlich wenn die Realität mal wieder nicht perfekt ist



# Wundervolle Dokumentation

➤ Um Cypress zu lernen

➤ Best Practices

➤ Um die CI aufzusetzen

➤ ...

The screenshot shows a browser window with the title "Best Practices | Cypress Documentation". The URL is "docs.cypress.io/guides/references/best-practices#\_do...". The page content includes a sidebar with the "cypress docs" logo and a search bar. The main content area lists three best practices:

1. Don't target elements based on CSS attributes such as: `id`, `class`
2. Don't target elements that may change their `textContent`
3. Add `data-*` attributes to make it easier to target elements

## How It Works

Given a button that we want to interact with:

```
<button  
  id="main"  
  class="btn btn-large"  
  name="submission"  
  role="button"  
  data-cy="submit"  
>  
  Submit  
</button>
```

Let's investigate how we could target it:

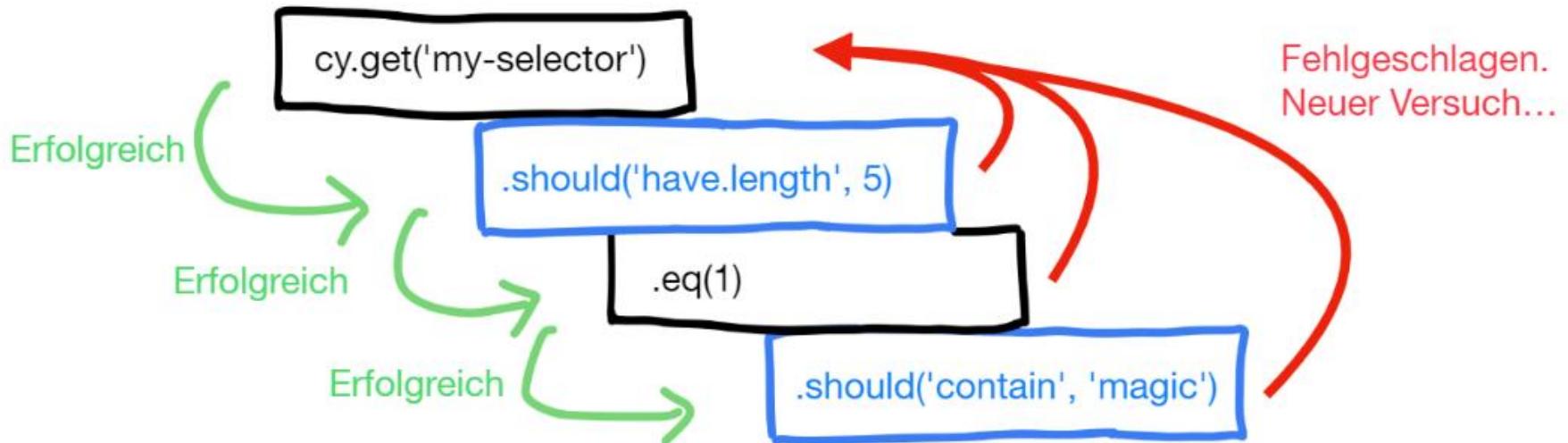
Selector	Recommended
<code>cy.get('button').click()</code>	<span style="color: red;">⚠ Never</span>

# Let's see the Magic

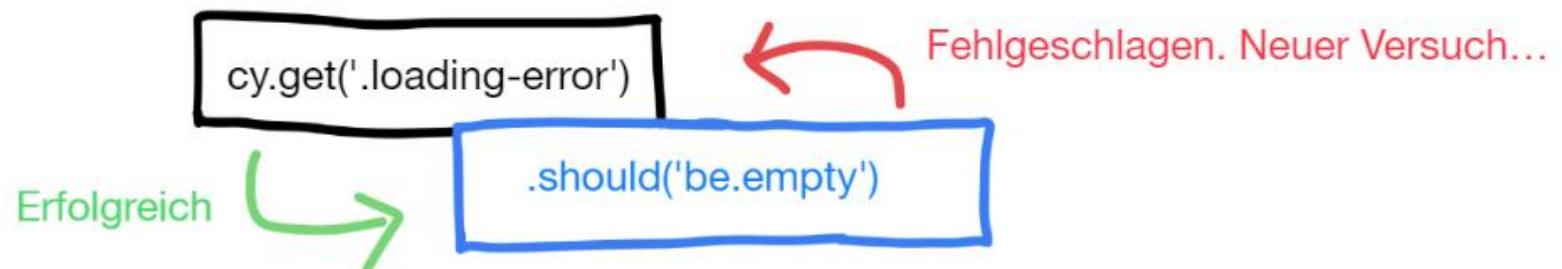
a

```
$ npm install --save-dev cypress  
$ npx cypress open
```

# Chain of Commands



Neue Chain



# Best Practices

## E2E Tests



# 7 Best Practices, die du im Hinterkopf behalten solltet



1.

Behalte die Perspektive  
des Endnutzers bei.



2.

## Priorisiere Testfälle

- Risk-Based Testing
- Smoke Testing).



3.

Halte deine Testumgebung  
so nah wie möglich an der  
Produktiv-Umgebung ohne  
die Produktiv-Umgebung  
zu stören!



4.

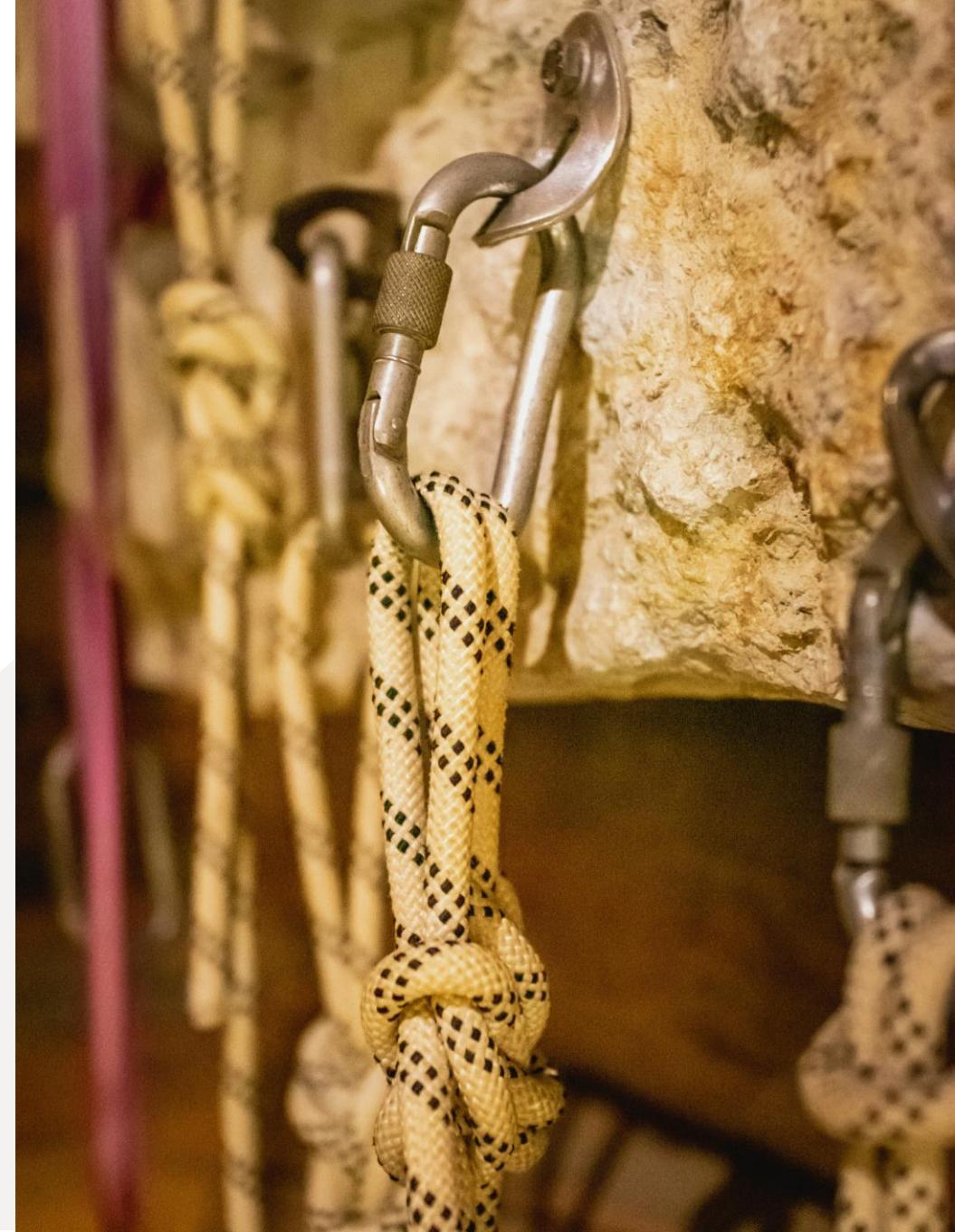
Halte deine Testdaten  
realistisch und aktuell.

Starte jeden Test mit frischen  
Daten, wenn möglich.



5.

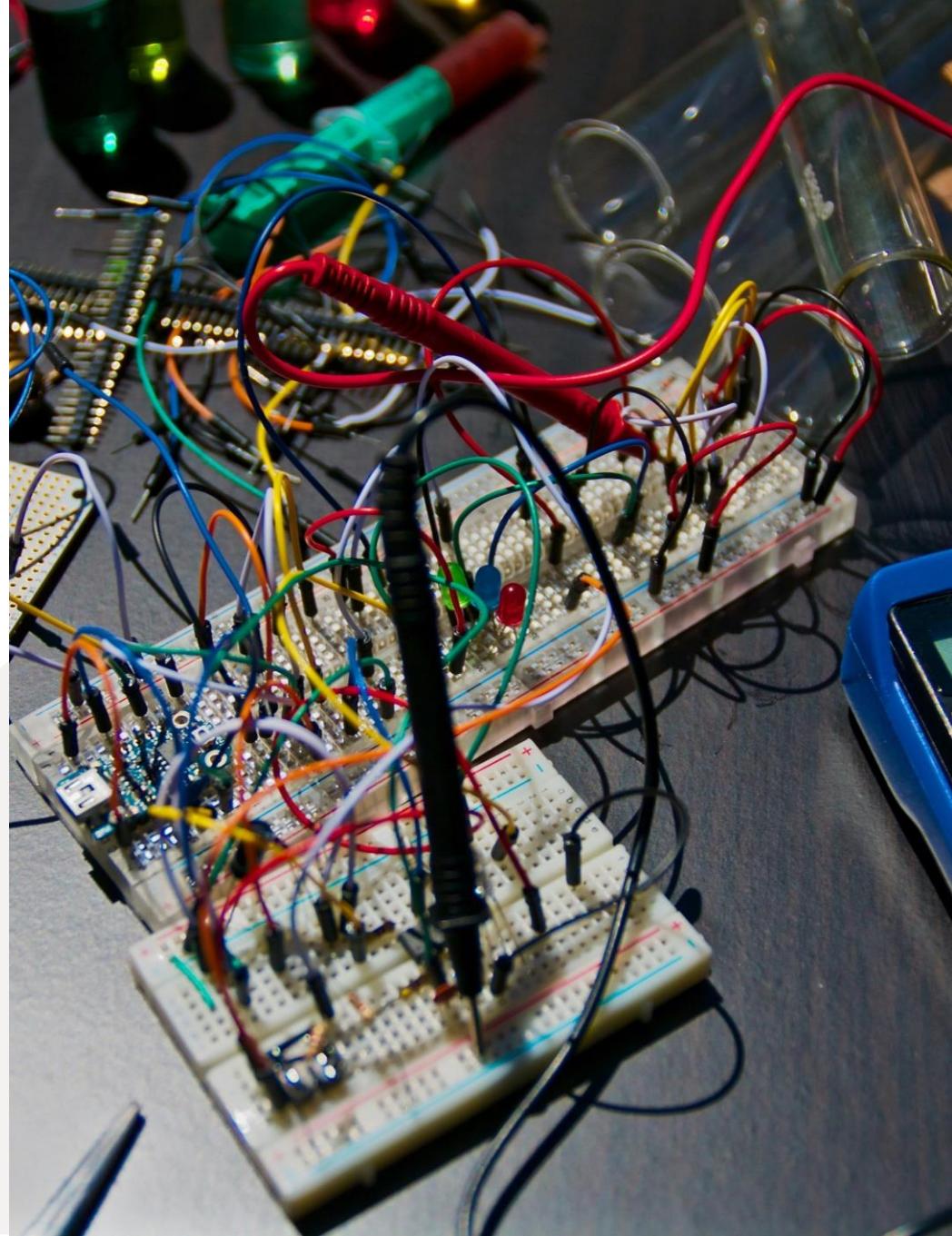
Nutze stabile Selektoren,  
um auf die UI-Elemente  
zuzugreifen.



6.

Halte Tests voneinander  
unabhängig.

→ Sie sollten sich nicht  
gegenseitig beeinflussen.



7.

Denk dran: Tests werden mit  
der Zeit zunehmend  
schwerer wartbar und  
zeitaufwändig.



# Zum Repo auf GitHub



[https://github.com/hannahebert/js\\_days\\_cypress](https://github.com/hannahebert/js_days_cypress)

