

Web Engineering

E2E-Tests, Validierung mit Spring

Adrian Herzog

(basierend auf der Arbeit von Michael Faes, Michael Heinrichs & Prof. Dierk König)

Rückblick: Templates mit Pebble

Contact List

[Home](#)[Contacts](#)[About](#)

Contacts

Mabel Guppy

Lauree Clouter

Aileen Cattrell

Bax McGrath

Graeme Impett

Moll Mullarkey

Charla Spinster

Seana Burberye

Hallsy Robertet

Kristofer Bril

Orel Faulconbridge

First name

Aileen

Last name

Cattrell

Email addresses

mstaves0@opensource.org

vmash1@patch.com

Phone numbers

525-477-4251

331-888-1254

657-694-9841

E2E-Tests

(End-To-End Tests)

Etwas Wichtiges vorweg

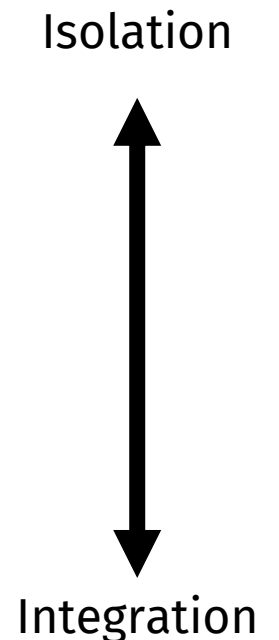
- Wir sprechen in diesem Kurs immer mal wieder von **«Tests»**, damit meinen wir genau genommen **automatisierte Checks**.
- Diese automatisierten Checks sind lange nicht alles, was es braucht, um **Software im eigentlichen Sinne des Wortes zu testen**.
- Es bleiben viele Risiko-Bereiche übrig, die **menschliches Testen** erfordern:
 - Sicherheit, Usability, User Experience, Darstellung, ...
 - Erfüllung der Anforderungen (auch der unausgesprochenen)
 - Lücken in der Test-Automatisierung (wir können unmöglich alles automatisiert prüfen)
 - Missverständnisse, Umsetzungsfehler (auch in «Tests»), ...
 - ...

aber sie bringen
mehr Stabilität

Einordnung

Typische Kategorien für *automatisierte* Tests:

Test-Art	Beschreibung
<i>Unit Test</i>	Testet eine einzige (kleinste) Einheit der Software, z. B. eine Klasse oder Methode.
<i>Integration Test</i>	Fügt mehrere Einheiten zusammen und testet sie als Gruppe. Soll Fehler bei der Interaktion von Einheiten aufdecken.
<i>E2E Test (System Test)</i>	Testet das System als Ganzes, aus Benutzer- bzw. Schnittstellensicht.

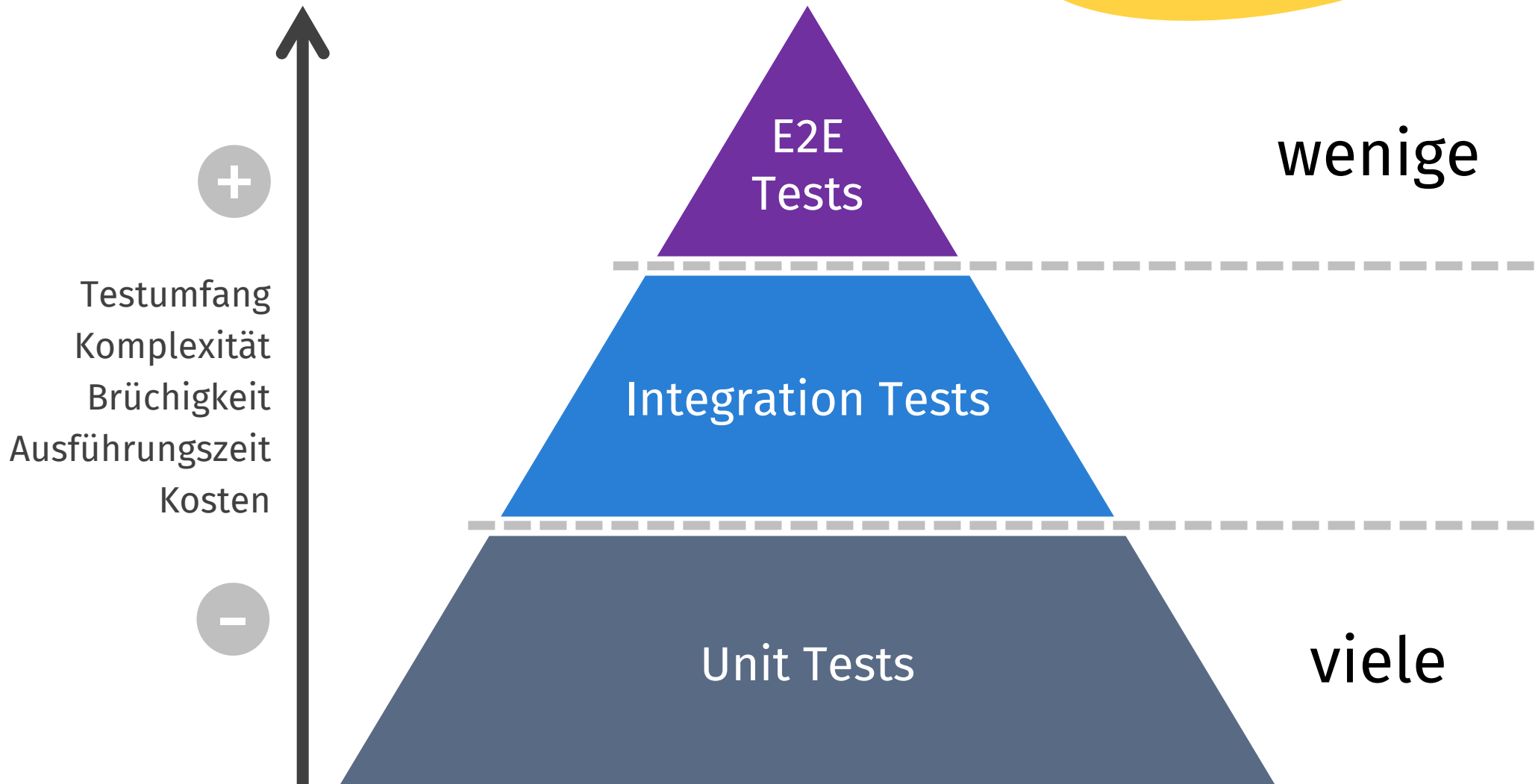


Was sind die «Ends» in E2E-Tests?

- E2E-Tests decken **die ganze Applikation** ab, von einem «End» (UI) über die Controller und die Applikationslogik bis zum anderen «End» (Datenbank).
 - **System-Test** ist auch ein treffender Begriff (wobei wie bei E2E definiert werden muss, was zum System gehört)
 - Der Begriff *UI-Test* ist zu unscharf (könnte auch nur das UI betreffen)
- Was ist mit weiteren Applikationen, die z.B. über Web Services angeschlossen sind?
 - Mocken, falls sie die Test-Stabilität beeinträchtigen oder die Kontrolle über die Testdaten erschweren.

Test Automation Pyramid

It's just a model!
"All models are wrong,
but some are useful"



Ziel von automatisierten E2E-Tests

~~Prüfen, ob die Applikation als Ganzes keine Fehler hat.~~

zu kompliziert, zu aufwändig,
praktisch unmöglich

*Prüfen, ob die allerwichtigsten Features mit typischen
Eingaben keine offensichtlichen Probleme haben.*

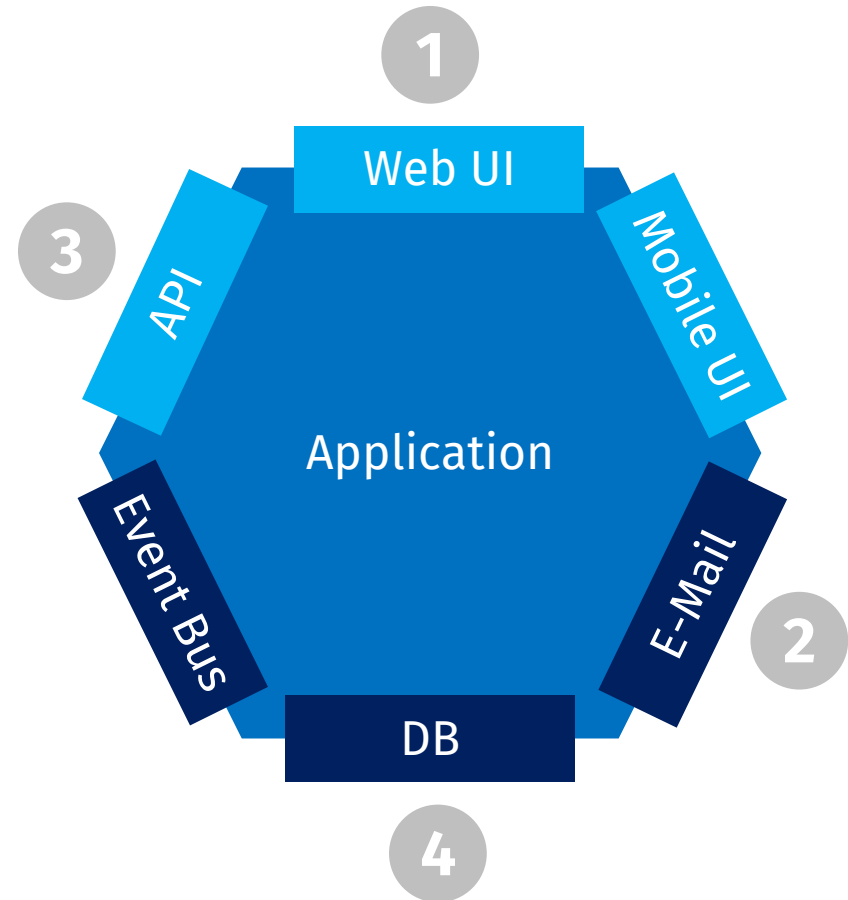
machbar und oft wertvoll

Gehen E2E-Tests immer über das GUI?

Nein, alle Schnittstellen können für E2E-Tests verwendet werden.

Beispiel:

- 1 Login über Web UI
- 2 Prüfen ob Mail verschickt wurde
- 3 Daten-Import über API
- 4 Prüfen was in der DB steht



Demo E2E-Test mit Selenium

```
driver.get("http://localhost:8080/convert");

driver.findElement(By.name("feet")).sendKeys("5");
driver.findElement(By.name("inches")).sendKeys("2");
driver.findElement(By.cssSelector("input[type=submit]"))
    .click();

var result = driver.findElement(
    By.cssSelector("[data-test-id=result]"))
    .getText();

assertEquals("157 cm, 5 mm", result);
```

Length converter

Input

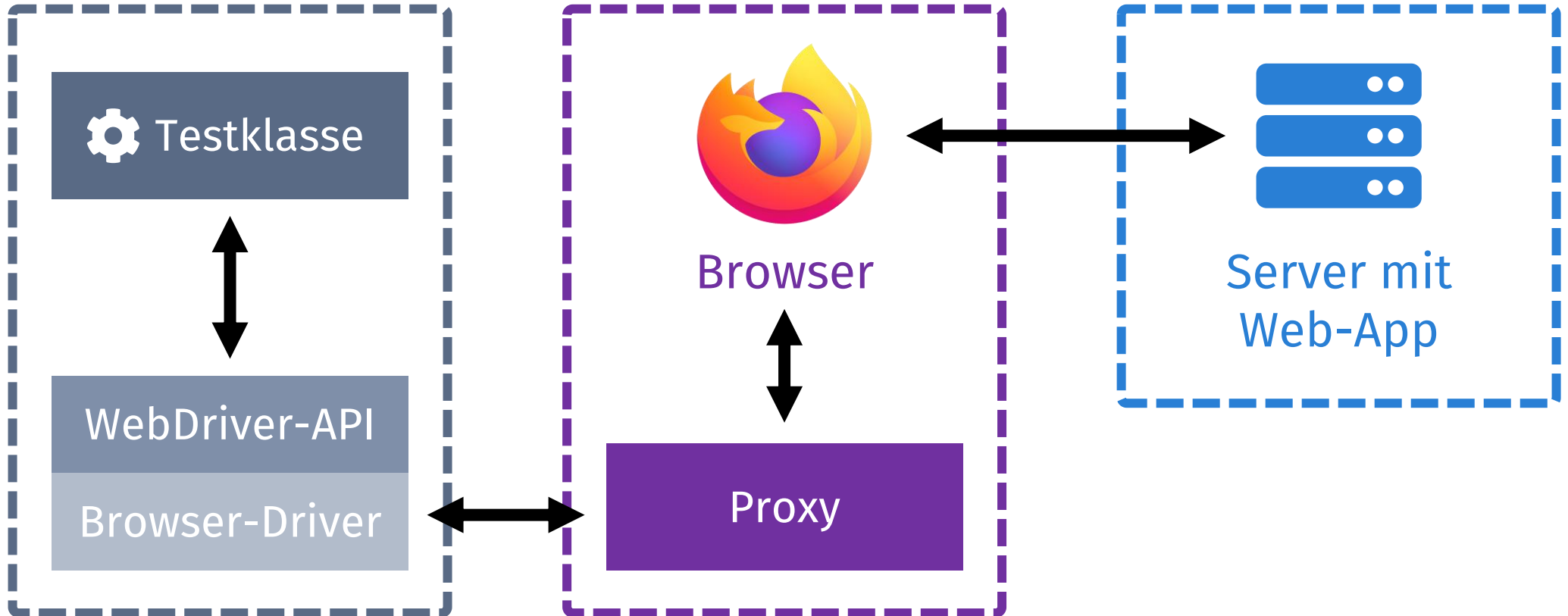
Feet

Inches

Result

157 cm, 5 mm

Selenium Setup



WebDriverManager

Immer manuell den korrekten Browser-Driver zu installieren ist eine ist fehleranfällig (Version muss zum Browser passen).

Deshalb gibt es den WebDriverManager, der sich darum kümmert:

```
✓ ConverterE2ETest 1 sec 866 ms
  ✓ e2eTest() 1 sec 866 ms
"C:\Program Files\Java\jdk-21.0.2\bin\java.exe" ...
20:15:38.348 [main] INFO io.github.bonigarcia.wdm.WebDriverManager -- Using chromedriver
129.0.6668.100 (resolved driver for Chrome 129)
20:15:38.375 [main] INFO io.github.bonigarcia.wdm.WebDriverManager -- Exporting webdriver
.chrome.driver as C:\Users\Adrian\.cache\selenium\chromedriver\win64\129.0.6668
.100\chromedriver.exe
```

```
<dependency>
  <groupId>io.github.bonigarcia</groupId>
  <artifactId>webdrivermanager</artifactId>
  <version>5.9.2</version>
  <scope>test</scope>
</dependency>
```

@SpringBootTest

E2E-Test setzen laufende Applikation, inkl. Web-Server voraus.

Wie starten wir Applikation *automatisch* vor Testausführung? Natürlich mit einer Annotation! ✨🔧

```
@SpringBootTest(webEnvironment = RANDOM_PORT)
class LengthConverterE2ETest {

    @LocalServerPort
    int port;

    @Test ...
}
```

Lädt gesamte Applikation (alle Beans) und startet lokalen Web-Server.

Übung 1: Einfacher E2E-Test


Erstelle einen einfachen E2E-Test, der testet, ob auf der Contacts-Seite alle 30 Kontakte-Links angezeigt werden. Weiter soll getestet werden, dass durch das Klicken auf einen solchen Link die Details des entsprechenden Kontakts angezeigt werden.

Schnelle und langsame Tests

E2E-Tests dauern deutlich länger als Unit-Tests. Unpraktisch, wenn man häufig testet.

Lösung: Maven unterteilt in «normale» (Unit-)Tests und «Integration-Tests» (alle langsamen, inkl. E2E).

```
@SpringBootTest(...)
class LengthConverterIT {
    ...
}
```



Tests mit **IT** im Namen werden nicht während `mvn test`-Phase, sondern nur während `mvn verify` ausgeführt. Braucht Plugin:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-failsafe-plugin</artifactId>
</plugin>
```

Übung 2: mvn verify

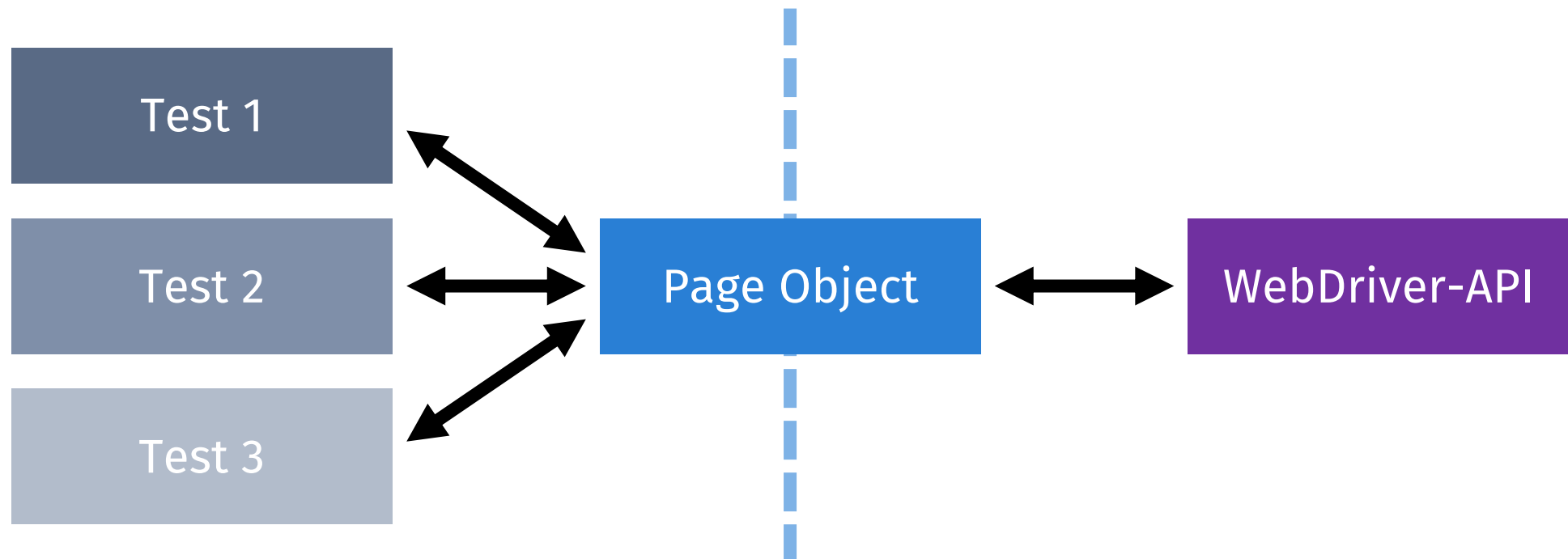
Passe das Projekt so an, dass `mvn test` nur noch die Unit Tests ausführt. Mit `mvn verify` sollen auch die E2E tests ausgeführt werden.

Achtung: Da wir den Maven Wrapper verwenden, heissen die Befehle leicht anders (z.B. `.\mvnw.cmd test` unter Windows).

Page-Object Pattern

Problem: UI-Tests sind stark an HTML-Details gebunden (IDs, CSS-Selektoren). Wenn Details ändern → alle Tests anpassen.

Lösung: HTML-Details in eigene *Page-Object*-Klasse abkapseln.



Page-Object: Definition

```
public class FormPage {  
  
    public FormPage(WebDriver driver, int port) {  
        driver.navigate().to("http://localhost:" + port + "/");  
        PageFactory.initElements(driver, this);  
    }  
  
    @FindBy(id = "inches")  
    private List<WebElement> inchesField;  
    @FindBy(css = "input[type=submit]")  
    private List<WebElement> submitButton;  
  
    public Optional<WebElement> getInchesField() {  
        return inchesField.stream().findFirst();  
    }  
  
    ...  
}
```

initialisiert
@FindBy-Attribute

Beispiel-
Verwendung

Page-Object: Verwendung

In eigentlicher Testklasse funktioniert Interaktion mit HTML-Inhalt ausschliesslich über Page-Objects:

```
WebDriver driver = new HtmlUnitDriver();  
var formPage = new FormPage(driver, port);  
  
assertTrue(formPage.getInchesField().isPresent());  
assertEquals("inches", formPage.getInchesField().get()  
    .getAttribute("name"));
```

Falls HTML-Struktur ändert, muss auch Code in `FormPage`-Klasse geändert werden. Aber nicht die (vielen) Tests!

Übung 3: Page-Object Pattern

Erstelle eine Page-Object Klasse für die Contacts-Seite, mit Attributen für die Kontakte-Links und die Kontakt-Details.

Refactore anschliessend den Test `ContactsPageIT`, sodass solche Page-Objects verwendet werden.

Spring Boot Konfiguration

Konfiguration

Spring bietet einfache Möglichkeit, um App zu konfigurieren:

```
server.port=7070
```

```
logging.level.ch...webec.lengthconverter=debug
```

```
logging.level.root=warn
```

```
my.own.property=foo
```

application.properties
(in resources)

Zugriff, z. B. in Controller:

```
@Controller
```

```
public class ConvertController {
```

```
    @Value("${my.own.property}")
```

```
    private String property;
```

```
    ...
```

Spring Boot Validierung

Validierungs-Optionen

Option	Ort	Beispiel
In Formular	Client	<code><input type="number" min="1"></code>
Data-Binding (deklarativ)	Server	<pre>@GetMapping(...) public String foo(@Min(1) int value){ ... }</pre> <div>«JSR 303»</div>
In Controller (imperativ)	Server	<pre>@GetMapping(...) public String foo(int value) { if (value < 1) { model.put("error", "Wert < 1"); } }</pre>

Validierungsfehler abfangen



```
@Controller
```

```
@Validated
```

```
public class ConverterController {
```

```
    @GetMapping("/convert")
```

```
    public String convert(@Min(0) int inches, Model model) {
```

```
        ...
```

```
    }
```

```
    @ExceptionHandler({ConstraintViolationException.class})
```

```
    @ResponseStatus(HttpStatus.BAD_REQUEST)
```

```
    public String invalidRequest(Model model) {
```

```
        model.addAttribute("error", "Inches muss pos. sein");
```

```
        return "contacts";
```

```
    }
```

```
}
```

```
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-validation</artifactId>
```

```
</dependency>
```

Validierung auf Objekten

```
@GetMapping("convert")
public String convert(@Valid ImperialLength input, Model m)
{
    ...
}
```

```
public class ImperialLength {

    @Min(0)
    private Double feet;

    @Min(0)
    @Max(12)
    private Double inches;

    ...
}
```

```
@ExceptionHandler(MethodArgumentNotValidException.class)
public String handleValidationException(MethodArgumentNotValidException ex, Model model) {
    List<ValidationError> errors = new LinkedList<>();

    for (FieldError error : ex.getBindingResult().getFieldErrors()) {
        errors.add(new ValidationError(error.getField(), error.getDefaultMessage()));
    }

    model.addAttribute("validationErrors", errors);
    model.addAttribute("input", ex.getBindingResult().getTarget()); // Preserve input

    return "converter";
}
```

Typumwandlung

Selbst ohne ausdrückliche Validierung: Typumwandlung

```
@Controller
public class ConverterController {

    @GetMapping("/convert")
    public String convert(int inches, Model model) {
        ...
    }

    @ExceptionHandler(MethodArgumentTypeMismatchException.class)
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    public String invalidRequest(Model model) {
        model.addAttribute("error", "Inches muss Zahl sein");
        return "contacts";
    }
}
```

Fragen?

