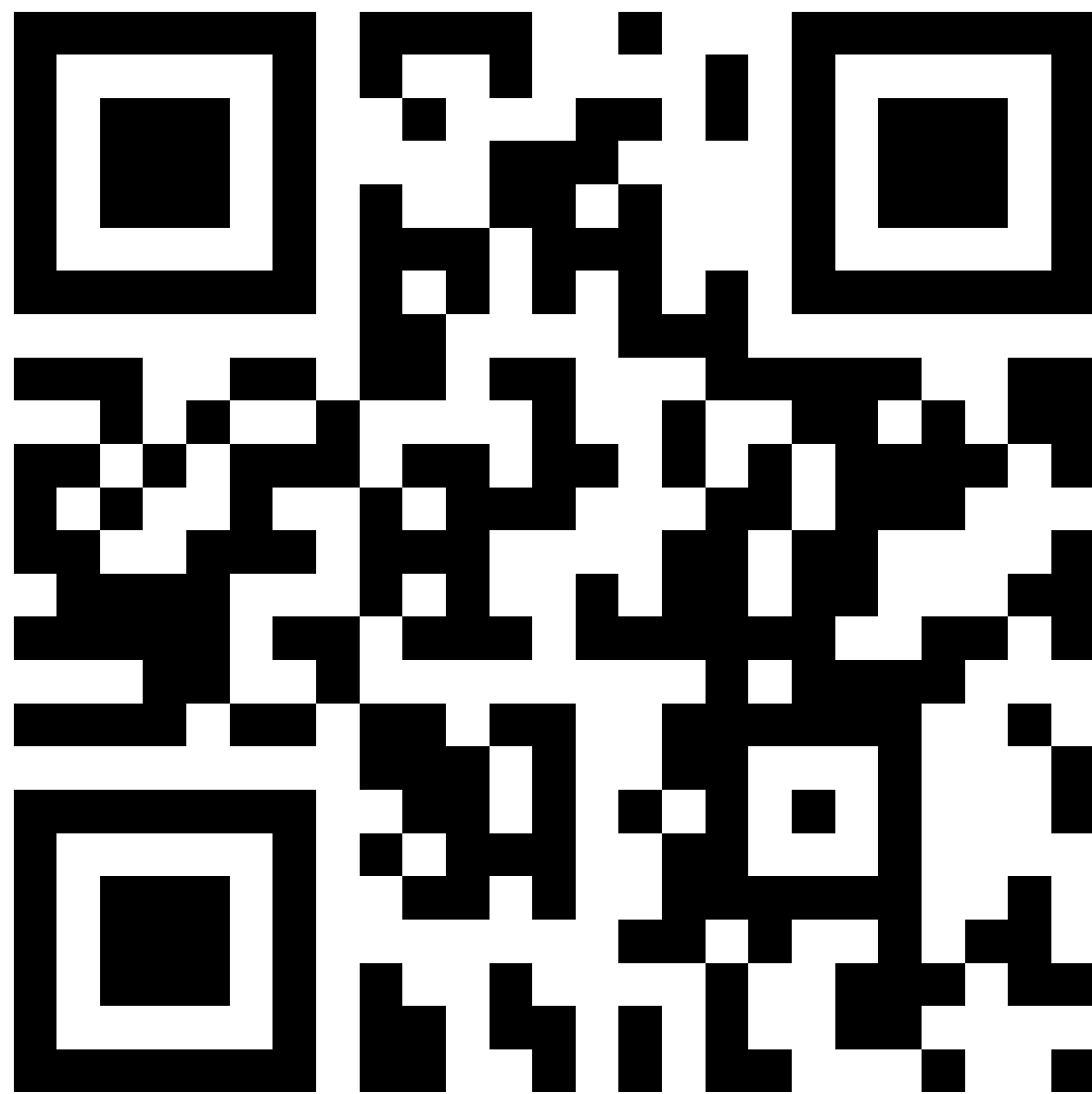


Web Engineering

DHBW Stuttgart · Informatik / Wirtschaftsinformatik

Sommersemester 2026 · Sitzung 7 — Testing

<https://librete.ch/dhbw/>



Ressourcen zum Selbstlernen

- **CODE CRISPIES** – codecrispi.es
- **Online Code-Editor** – codepen.io
- **MDN** (Mozilla Developer Network) – developer.mozilla.org/de/
- **Flexbox-Spiel** – flexboxfroggy.com
- **Grid-Spiel** – cssgridgarden.com

Testing

Unit • Integration • End-to-End

Inhalt

1. Test Automation Pyramid
2. Test Automation Trophy
3. Static Code Analysis
4. Unit Tests
5. Integration Tests
6. End-to-End Tests (API + Client)
7. Live Coding

Test Automation Pyramid

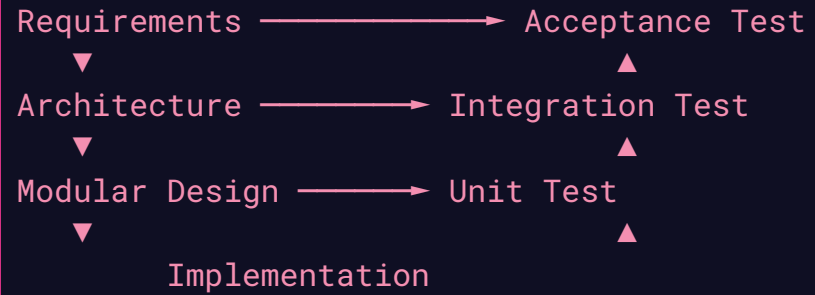
```

  ▲ manual / exploratory
  ▲ ▲ E2E (UI + API)
  ▲ ▲ ▲ Integration (class + real deps)
  ▲ ▲ ▲ ▲ Unit (class + mocked deps)
  ▲ ▲ ▲ ▲ ▲ Static analysis (compiler, linter)

```

- Oben: wenige, langsam, teuer, brüchig
- Unten: viele, schnell, billig, stabil

Test Automation V-Modell



Jede Design-Phase hat Testpendant. Testfälle früh ableiten.

Test Automation Trophy (Kent C. Dodds)

E2E	← klein
Integr.	← GROSS (Hauptfokus)
Unit	← mittel
Static	← Basis (TS, ESLint)

Andere Gewichtung: **Integration im Zentrum**, Static als breite Basis.

Testing Frameworks Übersicht

Static Code Analysis

- ESLint, Prettier, SonarQube, TypeScript

Unit + Integration

- jest, mocha, **vitest**

End-to-End

- **Cypress, Playwright**, supertest (API only)

Visual / Interaction

- Storybook, Chromatic

Static Code Analysis

- Läuft in Millisekunden
- Findet Syntaxfehler, ungenutzte Variablen, undefined refs
- Kein Runtime-Overhead
- **Erste Verteidigungslinie**

```
npm i -D eslint prettier  
npx eslint --init  
npx prettier --write .
```

SonarQube: zentraler Server, Code Smells, Security, Coverage.

Unit Tests – Eigenschaften

- **Isolation** – jede Dependency gemockt
- **Deterministisch** – gleicher Input → gleicher Output
- **Schnell** – kein I/O, kein Netzwerk, kein FS
- **Fokussiert** – eine Funktion pro Test

Unit Test – Beispiel (vitest)

```
// cache.unit.test.js
import { describe, it, expect } from "vitest";
import { Cache } from "../cache.js";

describe("Cache", () => {
  it("creates and retrieves data with custom key", () => {
    const cache = new Cache();
    const data = { user: "wolfgang", city: "stuttgart" };

    const key = cache.create(data, "custom-key");

    expect(key).toBe("custom-key");
    expect(cache.get("custom-key")).toEqual(data);
  });
});
```

Integration Tests – Eigenschaften

- **Reale Komponenten** – echte Cache + Express-Instanzen
- **Supertest-Magie** – simuliert HTTP ohne Server-Startup
- **In-Process** – alles im selben Node-Prozess
- **Kein Netzwerk** – keine TCP-Connections

→ teste Zusammenspiel mehrerer Module ohne Deployment.

Integration Test – Beispiel (supertest)

```
// app.integration.test.js
import request from "supertest";
import { app, cache } from "../app.js";

test("full CRUD lifecycle with real cache", async () => {
  const res = await request(app)
    .post("/api/entries")
    .send({ name: "integration-test", status: "active" });

  expect(res.status).toBe(201);
  const { key } = res.body;

  expect(cache.keys()).toContain(key);
});
```

End-to-End Tests (API)

- **Echter Server** – HTTP auf TCP-Port
- **Echtes Netzwerk** – `fetch` über localhost
- **Echte Umgebung** – CORS, Middleware-Stack
- **User-Perspektive** – exakt was Clients sehen

E2E API Test – Beispiel

```
// index.e2e.test.js
import { createServer } from "node:http";
import { app } from "./app.js";

let server;
const TEST_PORT = 8081;
const baseUrl = `http://localhost:${TEST_PORT}`;

beforeAll(async () => {
  server = createServer(app);
  await new Promise((r) => server.listen(TEST_PORT, r));
});

afterAll(() => server.close());

test("complete user journey", async () => {
  const res = await fetch(`${baseUrl}/api`, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ message: "e2e", timestamp: Date.now() }),
  });
  expect(res.status).toBe(201);
});
```

End-to-End Tests (Client)

- **Echter Browser** – Chromium / Firefox / WebKit
- **Echtes DOM** – Klicks, Tippen, Scrollen
- **Echtes Rendering** – CSS, async Operationen
- **Cross-Origin** – Client- und Server-Kommunikation

Tools: **Cypress, Playwright**

Cypress – Minimalbeispiel

```
// cypress/e2e/counter.cy.js
describe("Counter App", () => {
  it("increments on click", () => {
    cy.visit("http://localhost:3000");
    cy.contains("button", "Clicked 0 times").click();
    cy.contains("button", "Clicked 1 times");
  });
});
```

```
npx cypress open    # GUI
npx cypress run     # headless / CI
```

Playwright – Minimalbeispiel

```
// tests/counter.spec.js
import { test, expect } from "@playwright/test";

test("counter increments", async ({ page }) => {
  await page.goto("http://localhost:3000");
  await page.getByRole("button").click();
  await expect(page.getByRole("button")).toContainText("Clicked 1 times");
});
```

```
npx playwright test
npx playwright test --ui
```

Coverage Reports

```
npx vitest run --coverage
```

File	% Stmts	% Branch	% Funcs	% Lines
cache.js	95.2	80.0	100.0	95.2
app.js	88.7	75.0	90.0	88.7

- `lcov.info` → CI / SonarQube
- HTML Report → `coverage/index.html`

Best Practices

- **AAA:** Arrange · Act · Assert
- Tests sind **Dokumentation** – beschreibend benennen
- Ein Verhalten pro Test
- Keine Logik im Test (keine `if` / `for`)
- **Test Doubles:** stub, mock, spy, fake
- CI: Tests bei jedem Push

Live Coding

Repo: `node-cache-api`

```
git clone https://github.com/nextlevelshit/node-cache-api
cd node-cache-api
npm install
npm test          # unit + integration
npm run test:e2e  # e2e
```