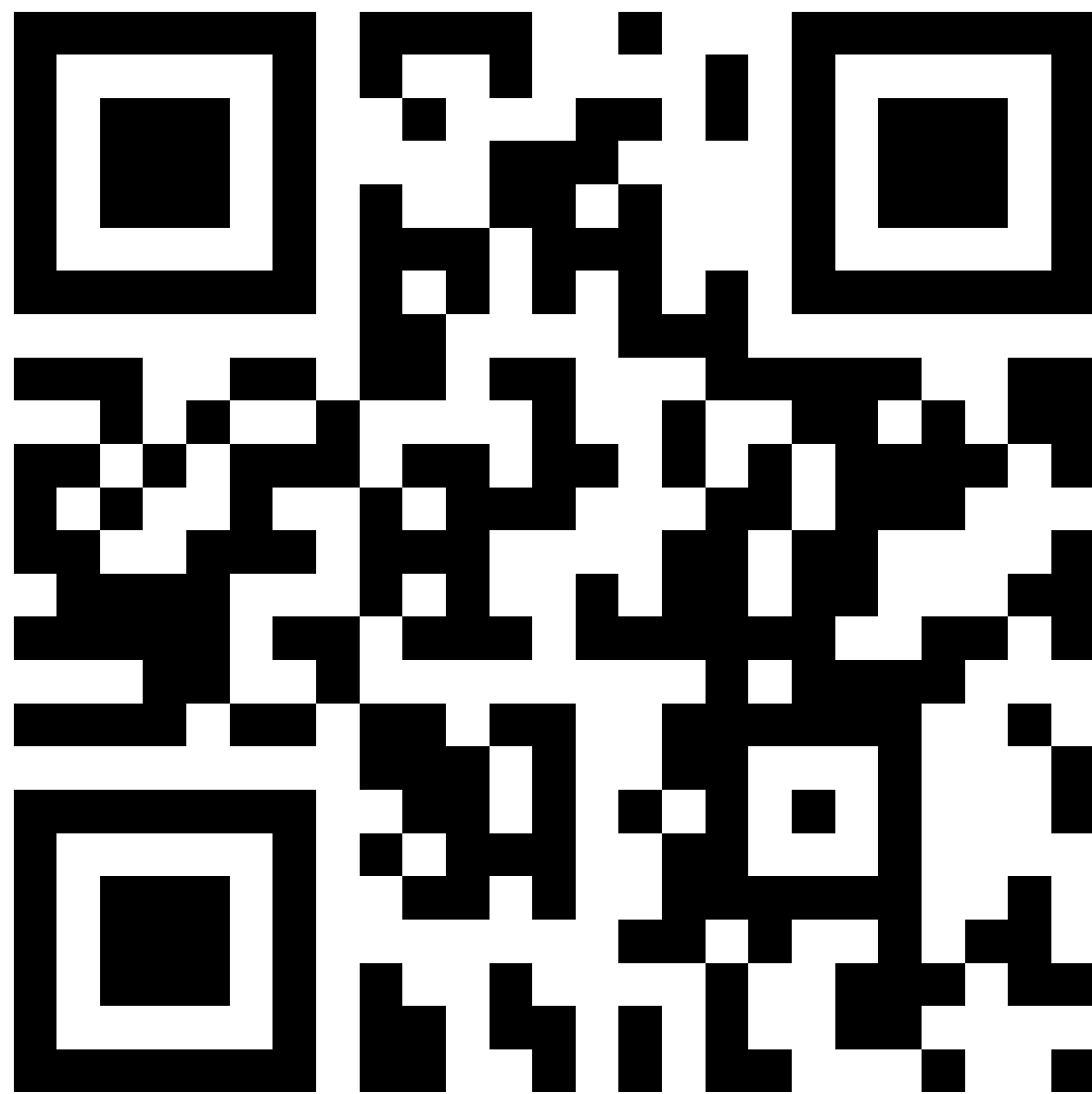


Web Engineering

DHBW Stuttgart · Informatik / Wirtschaftsinformatik

Sommersemester 2026 · Sitzung 10 – Wrap-up & Projektsupport

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



Best Practices

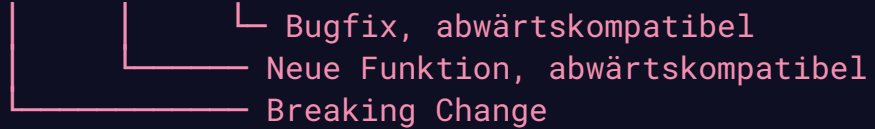
Wrap-up & Projektsupport

Inhalt

1. Semantic Versioning
2. Git Commit Messages (Udacity Style)
3. 12 Factor App
4. Make it Work · Make it Right · Make it Fast
5. Präsentations-Tipps
6. Feedback (Sandwich-Methode)
7. Prüfungsleistung Recap

Semantic Versioning (SemVer 2.0.0)

MAJOR.MINOR.PATCH



Beispiele:

- **1.4.2** → **1.4.3** Bugfix
- **1.4.3** → **1.5.0** neues Feature
- **1.5.0** → **2.0.0** Breaking Change

Pre-Release: **1.0.0-alpha.1**, **1.0.0-rc.2**

→ <https://semver.org/>

package.json: Versionsranges

```
{
  "dependencies": {
    "express": "^4.19.2",    // ≥4.19.2 < 5.0.0 (gleiche MAJOR)
    "lodash": "~4.17.21",   // ≥4.17.21 < 4.18.0 (gleiche MINOR)
    "react": "19.0.0"       // exakt
  }
}
```

`package-lock.json` pinnt die **tatsächlich installierten** Versionen.

Git Commit Messages – Udacity Style

```
<type>: <Subject>  
<blank>  
<Body>  
<blank>  
<Footer>
```

Type: `feat`, `fix`, `docs`, `style`, `refactor`, `test`, `chore`

Subject: Imperativ, ≤50 Zeichen, klein, kein Punkt

Body: **Was** und **Warum** – nicht **Wie**

Footer: Issue-Refs, Breaking Changes

Commit-Beispiele

Gut:

```
feat: add user logout endpoint
```

```
Allows authenticated clients to invalidate their session token  
on the server side instead of only deleting the cookie.
```

```
Closes #142
```

Schlecht:

```
fixed stuff  
WIP  
update  
asdf
```

→ <https://udacity.github.io/git-styleguide/>

Conventional Commits

```
feat(auth): add OAuth2 login  
fix(api): return 404 instead of 500 on missing user  
docs: update README install steps  
chore(deps): bump express from 4.18 to 4.19  
BREAKING CHANGE: drop Node 16 support
```

Vorteile: automatische Changelogs, automatische Version-Bumps (`semantic-release`).

12 Factor App

Methodik für moderne, skalierbare SaaS-Apps – <https://12factor.net/>

#	Faktor
1	Codebase – ein Repo, viele Deploys
2	Dependencies – explizit deklariert (<code>package.json</code>)
3	Config – via Environment-Variablen
4	Backing Services – als Attached Resources
5	Build, Release, Run – getrennt
6	Processes – stateless

12 Factor (2/2)

#	Faktor
7	Port Binding – self-contained, exportiert HTTP
8	Concurrency – horizontal skalieren via Prozesse
9	Disposability – schneller Start, Graceful Shutdown
10	Dev/Prod Parity – möglichst gleiche Umgebung
11	Logs – als Event-Streams (<code>stdout</code>)
12	Admin Processes – als One-Off

Config via ENV

```
// schlecht
const db = "postgres://user:secret@prod-db:5432/app";

// gut
const db = process.env.DATABASE_URL;
```

```
# .env (NICHT committen!)
DATABASE_URL=postgres://user:secret@db:5432/app
NODE_ENV=production
PORT=8080
```

`.env.example` mit Dummy-Werten committen.

Make It Work • Right • Fast

1. Make It Work → MVP, hässlich aber funktional
2. Make It Right → Refactor, Tests, Clean Code
3. Make It Fast → Profiling, Optimierung

Don't optimize what doesn't work.

Don't refactor what isn't working yet.

→ <https://wiki.c2.com/?MakeItWorkMakeItRightMakeItFast>

KISS • DRY • YAGNI • SOLID

- **KISS** – Keep It Simple, Stupid
- **DRY** – Don't Repeat Yourself
- **YAGNI** – You Aren't Gonna Need It
- **SOLID**:
 - **S**ingle Responsibility
 - **O**pen/Closed
 - **L**iskov Substitution
 - **I**nterface Segregation
 - **D**ependency Inversion

→ Faustregeln, keine Dogmen.

Präsentation – Aufbau

- **Einstieg** mit aktuellem Bezug (optional)
- **Problemdefinition**: welches Bedürfnis / welches Problem?
- Was hat **blockiert**, wie gelöst?
- **Geschichte mit Spannungsbogen**
- Vorher klären: Fragen während oder nach der Präsentation?

Präsentation – Pitfalls

- Am Thema vorbeireden
- Folien vorlesen
- Live-Demo ohne Backup-Video
- "Sieht man hier nicht so gut, aber..."
- Zu viel Text pro Folie
- Keine klare Botschaft pro Slide

→ **1 Folie = 1 Aussage.**

Feedback – Sandwich-Methode

1. **Eröffnung** – Lob / positive Beobachtungen
2. **Kern** – Kritik + konkreter Verbesserungsvorschlag
3. **Abschluss** – Lob / wertschätzender Ausklang

Pitfalls:

- Am Thema vorbeireden
- Bei Unklarheiten keine Fragen stellen
- Subjektives Feedback ("Farbe gefällt mir nicht")

→ **Konkret, sachlich, umsetzbar.**

Prüfungsleistung – Recap

Grundanforderungen (75 Punkte):

Punkte	Kriterium
20	Idee, Konzeption, Planung
5	Plattformunabhängigkeit
25	Clean Code (KISS, SOLID, DI, Testing, Error Handling)
15	Präsentation
10	Dokumentation (README, etc.)

Zusatzpunkte (max. 10): TypeScript, Docker, Dev/Prod-Parity, `.env`, npm-Publish, Domain, HTTPS, Responsive Design.

→ <https://librete.ch/DHBW/pruefungsleistung>

Abgabe + Termine

- **Code-Upload:** bis 27.07.
- **Präsentation:** 17.07. (in Gruppen, ~10 Min)
 - Pitch (~1 Min)
 - Agenda
 - Idee / Problem
 - Learnings & Pitfalls
 - "It is what it is" – Doku

70 Things I Regret as a Developer (Auswahl)

- Nicht früher Tests geschrieben zu haben
- Zu viel Zeit mit "perfektem" Setup statt Code
- Nicht öfter `git commit`
- Nicht früher in Pull Requests reviewt
- Keine Doku gepflegt
- Eigene Lösungen für Standard-Probleme
- Frameworks gehyped, Grundlagen vernachlässigt
- "Funktioniert bei mir" akzeptiert

→ Lehre: **Disziplin > Talent.**

Weiterlesen

- <https://12factor.net/>
- <https://semver.org/>
- <https://udacity.github.io/git-styleguide/>
- <https://www.conventionalcommits.org/>
- <https://refactoring.guru/>
- <https://martinfowler.com/>
- MDN Web Docs · web.dev · web.dev/learn

Viel Erfolg mit den Projekten!

Fragen? Probleme? → Telegram, Mail, Sprechstunde.