Workshop

CSS besser strukturieren mit BEM

Auch in großen Projekten mit CSS strukturiert,
berechenbar und flexibel arbeiten

Vorstellung

Andreas Gartz
Projektleiter / Product Owner
a.gartz@constructiva.de

constructiva SOLUTIONS

Constructiva Solutions GmbH
Irmintrudisstraße 13
53111 Bonn

Inhalt des Workshops

1. **Motivation**

   Was ist das Problem?

2. **Kurze Einführung in die Methode BEM**

3. **Hauptteil:** BEM praktisch erproben

   Hands-On!

   Take-Aways?

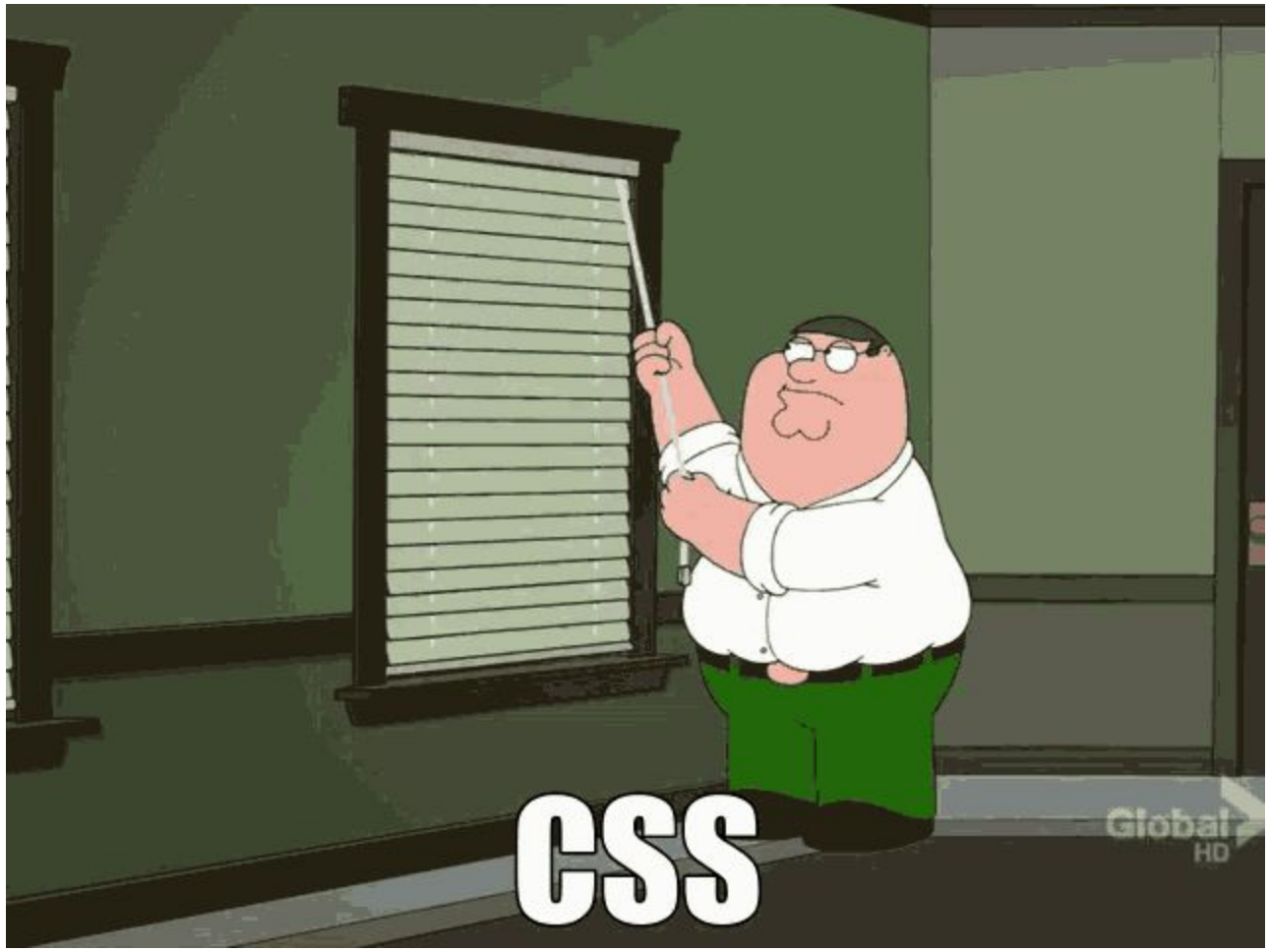
   Alternative Methoden?

4. **Zusammenfassung**

1. Motivation

Was ist das Problem?

Was ist das Problem?



Was ist das Problem?

```
/* Ausgangslage O_o! */
header > ul { … } /* breite Wirkung */
.main-content div.article.highlighted { … } /* spezifischer Selektor */
.nav.navbar_nav .language ul a { … } /* Framework überschreiben */
#home .nav.navbar_nav ul a { … } /* Id gibt scope vor */
```

Was ist das Problem?

```
/* Ausgangslage O_o! */
header > ul { … } /* breite Wirkung */
.main-content div.article.highlighted { … } /* spezifischer
.nav.navbar_nav .language ul a { … } /* Framework überschrei
#home .nav.navbar_nav ul a { … } /* Id gibt scope vor */
```

Was ist das Problem?

```
/* Ausgangslage O_o! */
header > ul { … } /* breite Wirkung */
.main-content div.article.highlighted { … } /* spezifischer
.nav.navbar_nav .language ul a { … } /* Framework überschrei
#home .nav.navbar_nav ul a { … } /* Id gibt scope vor */
body.home #sidebar .header ul { } /* <- das ändern */
```

Was ist das Problem?

```
/* Ausgangslage O_o! */

header > ul { … } /* breite Wirkung */

.main-content div.article.highlighted { … } /* spezifischer

.nav.navbar_nav .language ul a { … } /* Framework überschrei

#home .nav.navbar_nav ul a { … } /* Id gibt scope vor */

body.home #sidebar .header ul { } /* <- das ändern */


/* wir werden spezifischer…  O___o */
body.home #sidebar > div.header ul:first-child { }
```

Was ist das Problem?

```
/* Ausgangslage O_o! */
header > ul { … } /* breite Wirkung */
.main-content div.article.highlighted { … } /* spezifischer
.nav.navbar_nav .language ul a { … } /* Framework überschrei
#home .nav.navbar_nav ul a { … } /* Id gibt scope vor */
body.home #sidebar .header ul { } /* <- das ändern */

/* wir werden spezifischer…  O___o */
body.home #sidebar > div.header ul:first-child { }

/* getting the job done…  X___x */
body.home #sidebar > div.header ul:first-child { … !important; }
```

Was ist das Problem?

```
/* Ausgangslage O_o! */
header > ul { … } /* breite Wirkung */
.main-content div.article.highlighted { … } /* spezifischer
.nav.navbar_nav .language ul a { … } /* Framework überschrei
#home .nav.navbar_nav ul a { … } /* Id gibt scope vor */
body.home #sidebar .header ul { } /* <- das ändern */

/* wir werden spezifischer…  O___o */
body.home #sidebar > div.header ul:first-child { }

/* getting the job done…  X___x */
body.home #sidebar > div.header ul:first-child { … !important; }
```

Was ist das Problem?

CSS kann in größeren Projekten schnell chaotisch werden

und schwer zu verwalten sein

- Keine feste Namenskonvention: “Jeder kocht sein eigenes Süppchen”
- Unscharfe Benennung von Klassen, unklarer Einsatz
- Klassen werden breit eingesetzt und kombiniert

  mit gegenseitigen Beeinflussungen
- Die Vermeidung oder Behebung von Nebeneffekten führt oft zu noch

  längeren Selektoren, Nutzung von IDs und weiteren Unsauberkeiten

2. Kurze Einführung in die Methode BEM

Was ist BEM?

- BEM steht für "Block, Element, Modifier".
- Es ist eine Namenskonvention zur Benennung von CSS-Klassen und Elementen.
- BEM organisiert CSS in klar definierte Blöcke, Unterelemente und Varianten.

Die BEM-Struktur

- **Block**

  Eine unabhängige, wiederverwendbare Komponente

  (z. B. button, card, footer)

- **Element**

  Ein Teil eines Blocks, das nicht ausserhalb des Blocks genutzt werden darf

  (z. B. card__title, card__body, card__image)

- **Modifier**

  Eine Variante eines Blocks oder Elements

  (z. B. large, primary, featured, active, disabled)

Beispiele für das Namensschema

```
/* Block */
.button {}

/* Element */
.button__text {}

/* Modifier */
.button--primary {}
.button--disabled {}
```

- **Block**

  Eine unabhängige, wiederverwendbare Komponente

- **Element**

  Ein Teil eines Blocks, das nicht ausserhalb genutzt wird

- **Modifier**

  Eine Variante eines Blocks oder Elements

Beispiele für das Namensschema

```
/* Block */
.button {}

/* Element */
.button__text {}

/* Modifier */
.button--primary {}
.button--disabled {}
```

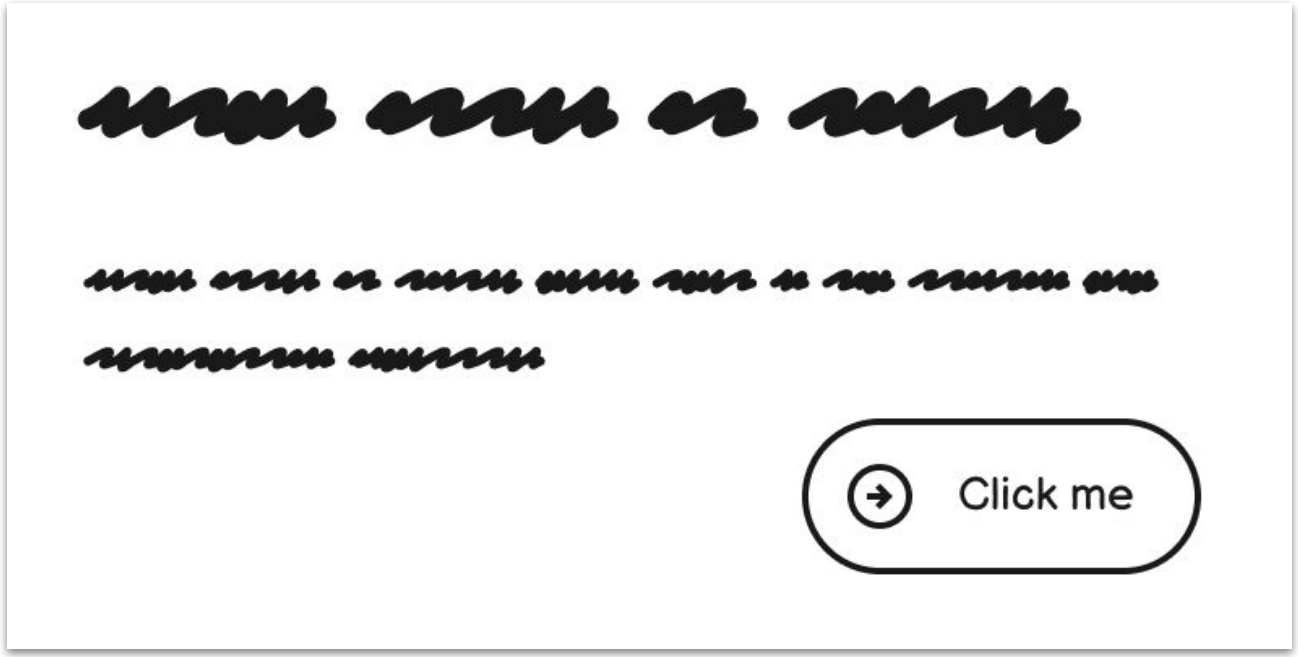
```
<button class="button">
    <span class="button__text">
        Click me
    </span>
</button>
```

Beispiele für das Namensschema

```css
/* Block */
.button {}

/* Element */
.button__text {}

/* Modifier */
.button--primary {}
.button--disabled {}
```

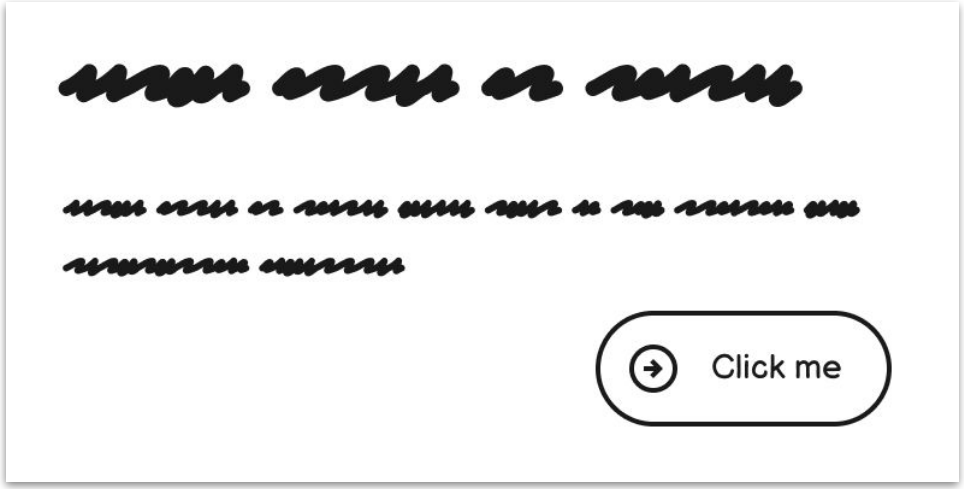
```html
<button class="button">
    <span class="button__text">…</span>
</button>

<button class="button button--primary">
    <span class="button__text">…</span>
</button>
```

Ein weiteres Beispiel



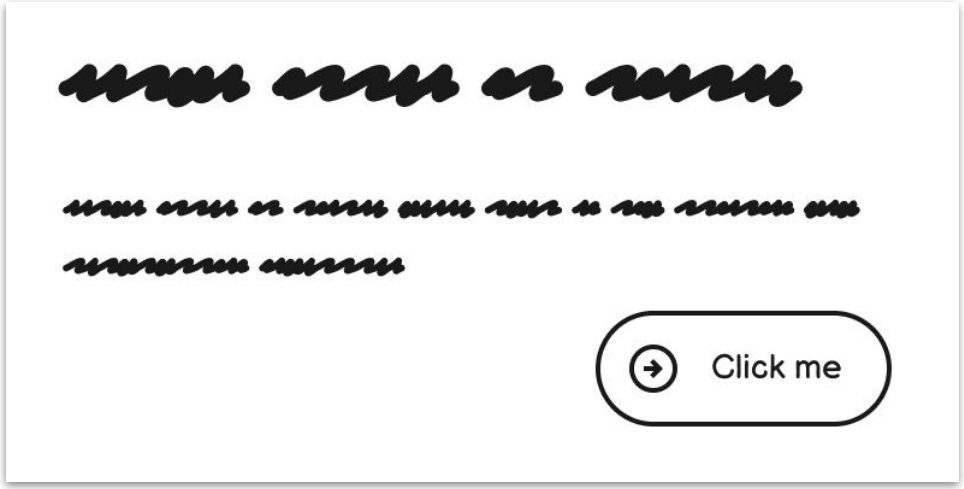

```
/* Block */

.card {}
```

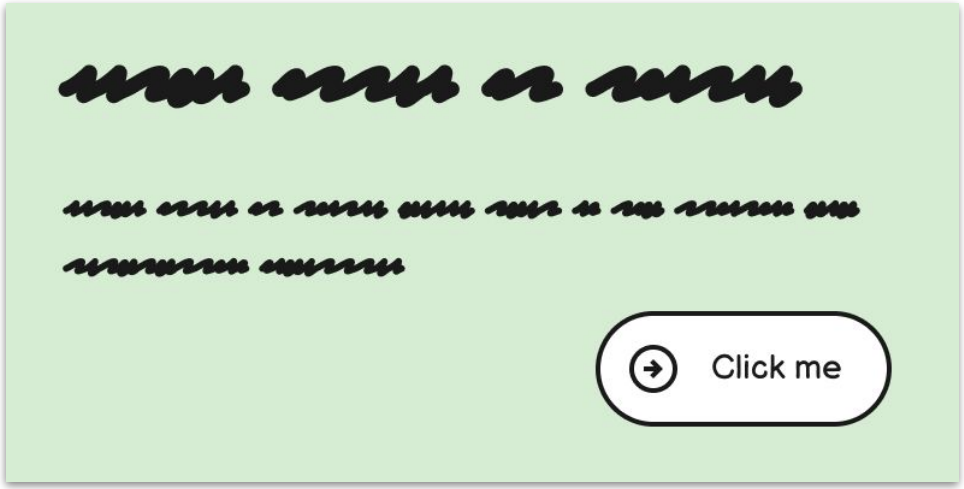
Ein weiteres Beispiel




```
/* Block */
.card {}

/* Elements */
.card__title {}
.card__text {}
.card__button {}
.card__button-icon {}
```

Ein weiteres Beispiel



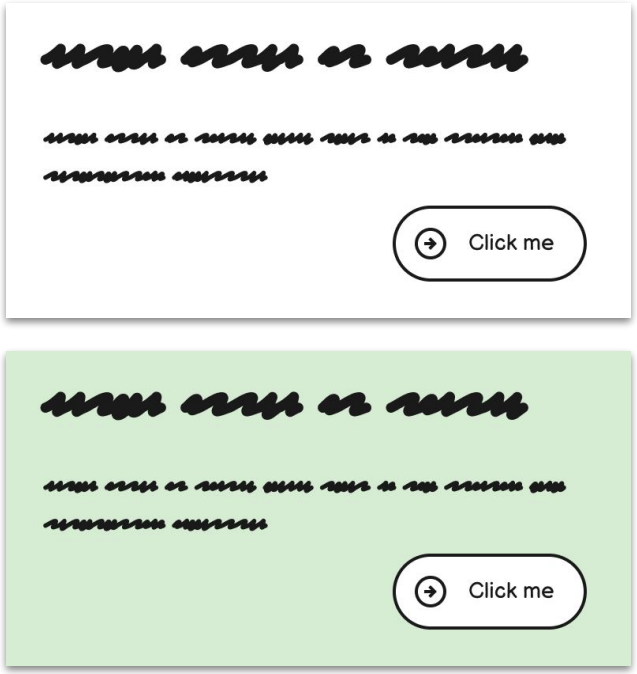


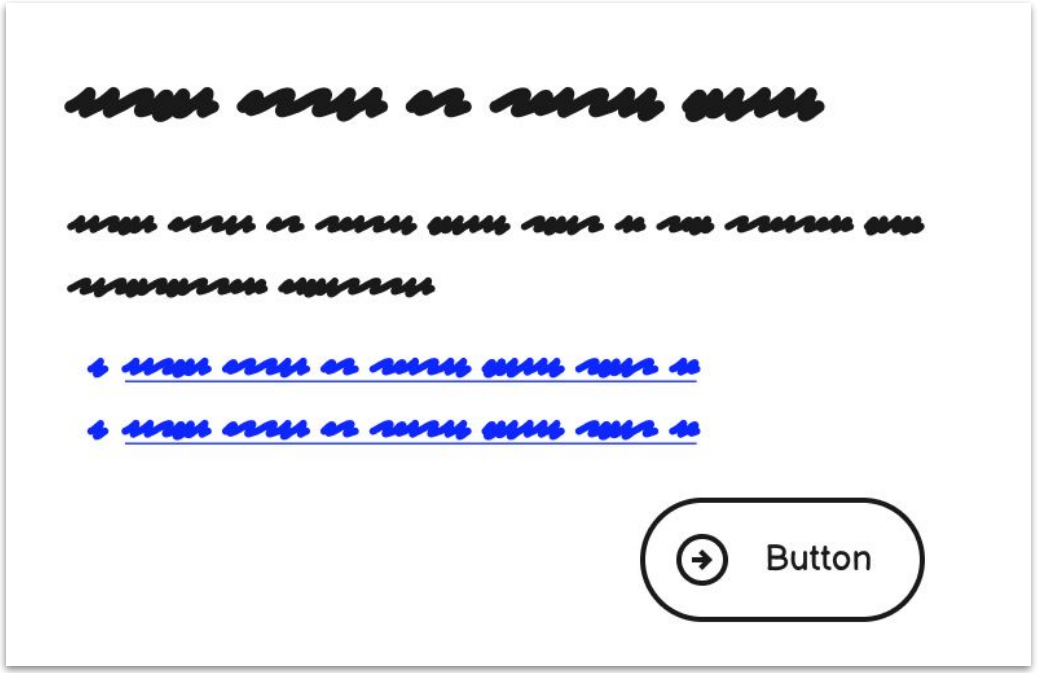

```
/* Block */
.card {}

/* Elements */
.card__title {}
.card__text {}
.card__button {}
.card__button-icon {}

/* Modifier */
.card--featured {}
```

3. BEM praktisch erproben

BEM praktisch erproben

Wir bauen eine erste Komponente!





https://tinyurl.com/bem-css | https://tinyurl.com/bem-css-git

BEM praktisch erproben

Take-Aways

- **Keine** “Enkel-Elemente” (card__list__item)
- Blöcke können problemlos geschachtelt in Blöcke eingebaut werden
- Selektoren immer mit nur einer Klasse schreiben. Ausnahme: Modifier.
- **Keine** Block-*übergreifenden* Styles umsetzen (wie .card .button {})

  -> lieber ein *wrapper-Element* nutzen (wie .card__button)
- Neue Blöcke erst schneiden, wenn eine sinnvolle Wiederverwendung eintritt

  oder die Tiefe eines Blocks zu unübersichtlich wird
- Jedes Team braucht seine festen Namenskonventionen!

  Diese müssen erarbeitet und besprochen werden!

Was macht BEM besser?

```
/* Ausgangslage O_o! */
.main-content div.article.highlighted { … }
.nav.navbar_nav .language ul a { … }
#home .nav.navbar_nav ul a { … }
body.home #sidebar .header ul { }
header > ul { … }
```

```
/* Zielbild */
.article--highlighted { … }
.navbar__language-link { … }
.navbar--home .navbar__link { … }
.header--sidebar .header__list { }
.header__list { … }
```

- klare Zuordnung & Einsatzbereich von Klassen (Namespace)
- geringe Spezifität in den Selektoren: 1 bis maximal 2 Klassen

Was macht BEM besser?

```
/* Ausgangslage O_o! */
.main-content div.article.highlighted { … }
.nav.navbar_nav .language ul a { … }
#home .nav.navbar_nav ul a { … }
body.home #sidebar .header ul { }
header > ul { … }
```

```
/* Zielbild */
.article--highlighted { … }
.navbar__language-link { … }
.navbar--home .navbar__link { … }
.header--sidebar .header__list { }
.header__list { … }
```

- das Team zieht an einem Strang (Namenskonvention)
- Einarbeitung und Erweiterung wird stark vereinfacht

Alternative Methoden

Ohne Anspruch auf Vollständigkeit:

- **OOCSS** (Object-Oriented CSS)

  `<div class="list list--header list--primary">...</div>`

- **SMACSS** (Scalable and Modular Architecture for CSS)

  `<div class="stage stage-narrow is-featured l-width-25 theme-dark">...</div>`

- **Atomic CSS**

  `<div class="mb-4 p-3 bg-blue text-white">...</div>`

- **Tailwind CSS** & Co.

  `<div class="bg-blue-500 text-white p-4">...</div>`

- **Styled Components** (z. B. in React, Vue ...)

4. Zusammenfassung

Wann eignet sich BEM als Methode?

- Langlebige Projekte, in denen Sauberkeit und Wartbarkeit ein Faktor sind
- CSS wird selbst strukturiert, Komponenten-basierte Arbeitsweise/Struktur
- Es wird bestenfalls kein CSS-Frameworks und keine CSS-Bibliotheken verwendet

Zusammenfassung

- BEM CSS ist eine leistungsstarke Methode zur Strukturierung von CSS
- Das Namespacing macht die Wartung die Skalierung einfach
- Geringe Spezifität von Selektoren verhindert Seiteneffekte
- BEM CSS fördert die Kommunikation (auch mit Designern und Kunden) und verbessert die Zusammenarbeit in größeren Entwicklerteams
- BEM muss von allen Teammitgliedern verstanden und gelebt werden. Konventionen müssen definiert werden. Das erfordert (anfangs) viel Kommunikation.

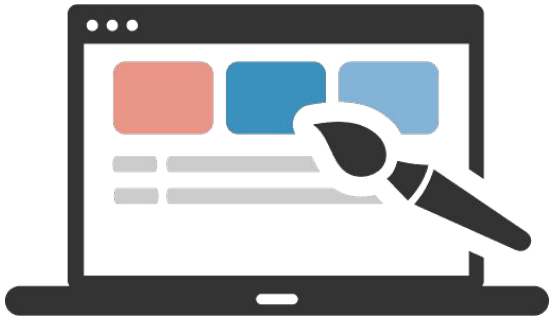
Kontakt

Andreas Gartz
Projektleiter / Product Owner
a.gartz@constructiva.de

www.constructiva.de

vcard

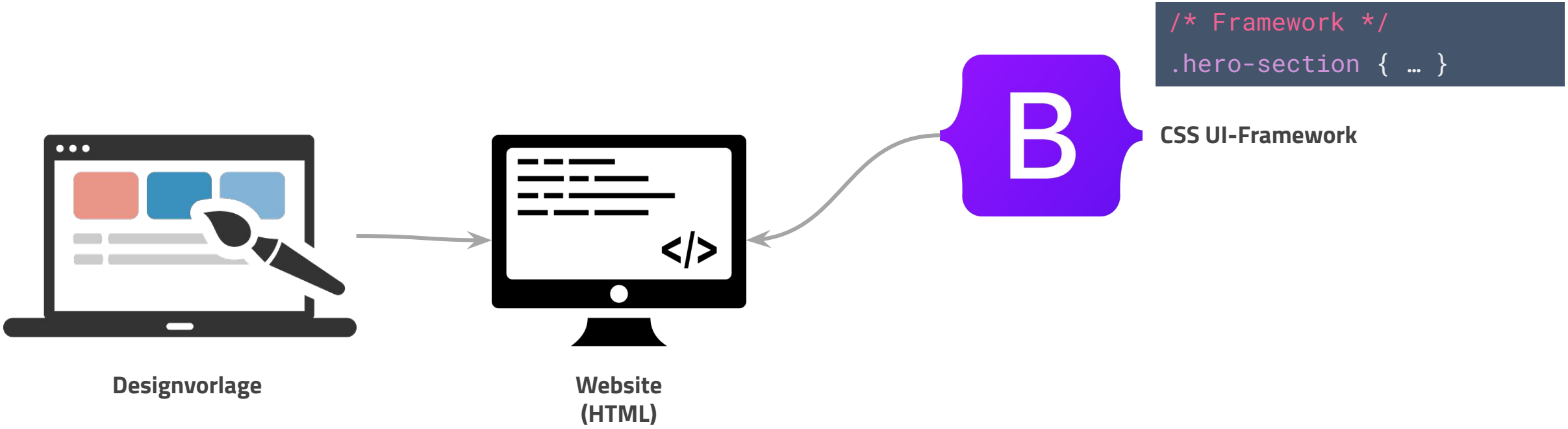
Was ist das Problem?



Designvorlage

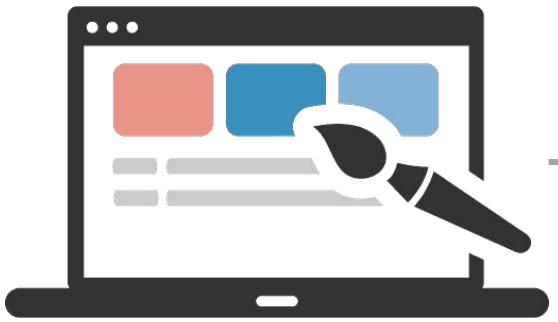
Designer, Kunde:
*“**Startseite mit Bühne**”*

Was ist das Problem?



Designer, Kunde:
*“**Startseite mit Bühne**”*

Was ist das Problem?



Designvorlage

Website
(HTML)

```
/* Framework */
.hero-section { … }
```

CSS UI-Framework

Custom CSS

```
/* Custom CSS - overwrite */
.hero-section.stage.home { … }
#home-stage.hero-section { … }
```

Designer, Kunde:
*“**Startseite mit Bühne**”*

Template /
View

Was ist das Problem?

* Don’t repeat yourself

Was können wir besser machen?

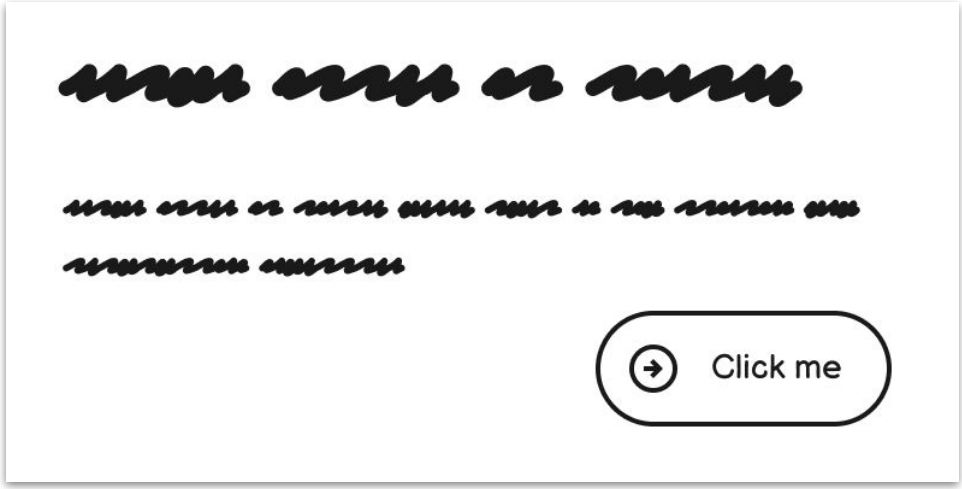
DRY*

KISS**

* Don’t repeat yourself
** Keep it simple, stupid

Fazit

Ein weiteres Beispiel




```
<article class="card">
    <h2 class="card__title">…</h2>
    <p class="card__text">…</p>
    <button class="card__button">
    …</button>
</article>
```

```
/* Block */
.card {}

/* Elements */
.card__title {}
.card__text {}
.card__button {}
.card__button-icon {}

/* Modifier */
.card--featured {}
```

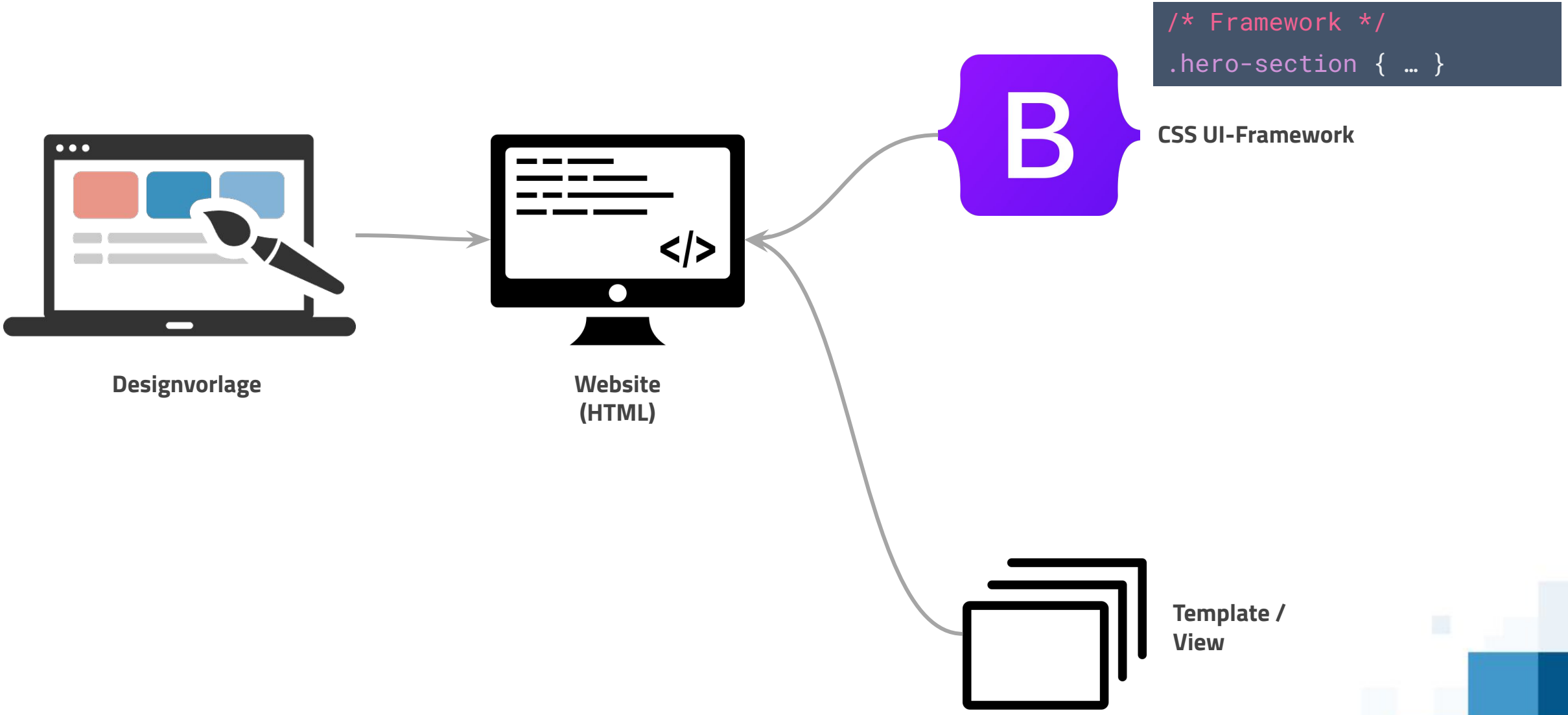
Was ist das Problem?

```
/* hochspezifisch
-> Wartung/Erweiterung schwierig */
.main-content div.article.highlighted { … }
#nav .language ul a { … }
.home #sidebar div.header ul { … }
```

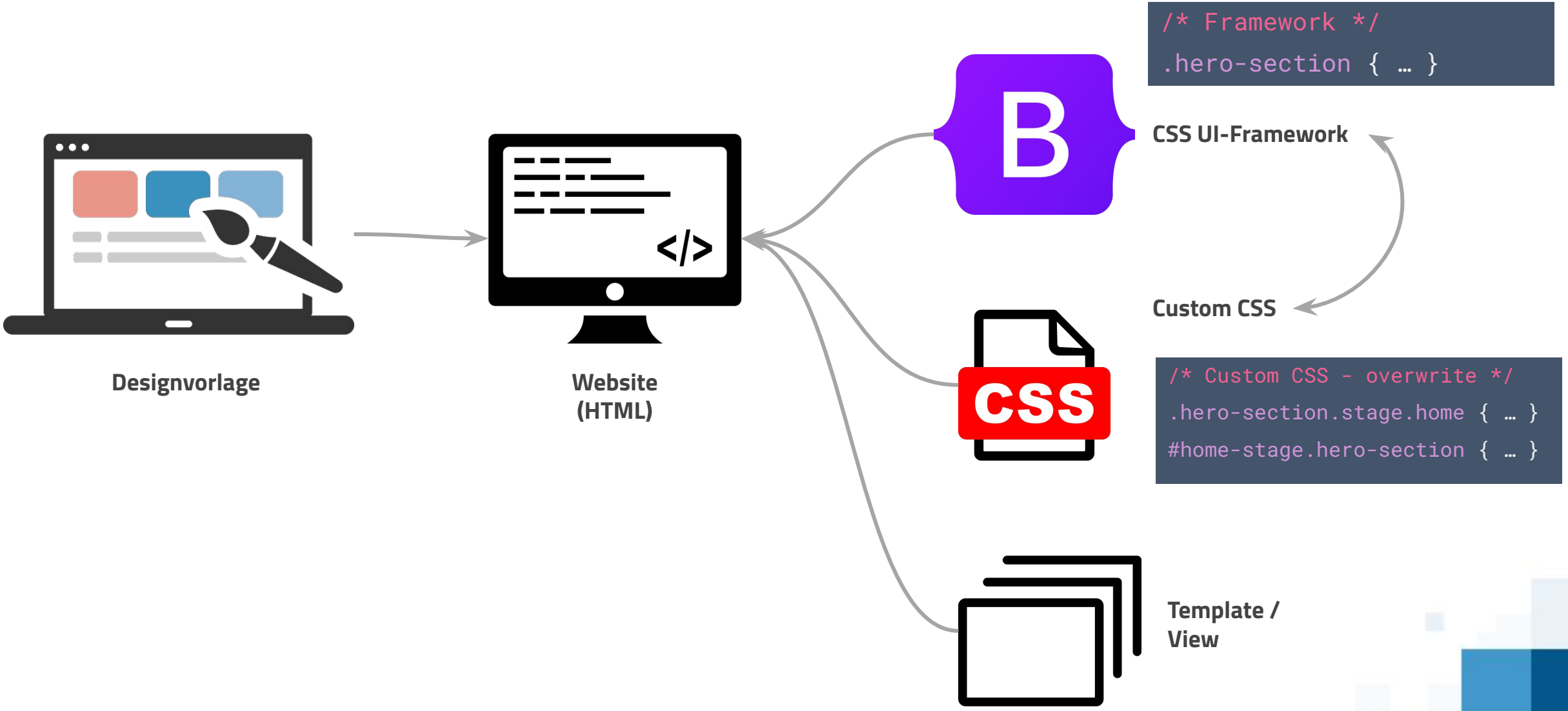
Was ist das Problem?

```
/* hochspezifisch
-> Wartung/Erweiterung schwierig */
.main-content div.article.highlighted { … }
#nav .language ul a { … }
.home #sidebar div.header ul { … }


/* wenig spezifisch -> viele Nebeneffekte */
header > ul { … }
```

Was ist das Problem?



Was ist das Problem?



Zweite Übung

Wir strukturieren die FrOSCon-Website mit BEM!

https://tiny.cc/bem-css-2

Ein persönlicher Rat

weniger

DRY*

MEHR

KISS**

Und das **Refactoring** nicht vergessen!

* Don’t repeat yourself
** Keep it simple, stupid!