

Code Reviews

Dozent: Prof. Dr. Michael Eichberg
Kontakt: michael.eichberg@dhbw.de
Version: 0.2

Folien: [HTML] <https://delors.github.io/lab-codereviews/folien.de.rst.html>
[PDF] <https://delors.github.io/lab-codereviews/folien.de.rst.html.pdf>
Fehler melden: <https://github.com/Delors/delors.github.io/issues>

Was sind Code Reviews

Definition

Bei einem Code Review geben Entwickler Feedback zu (fremden) Code; typischerweise vor einem Merge.

Code Reviews helfen dabei Fehler frühzeitig zu finden, die Qualität des Codes zu verbessern und sicherzustellen, dass alle Teammitglieder mit dem Code vertraut sind und verstehen wie der Code funktioniert.

Zweck von Code Reviews

- Qualitätssicherung
- Frühzeitiger Fehlererkennung
- Wissensverbreitung (im Team)

Best Practices

für Reviewer

- konstruktiv und respektvoll bleiben
- (kleine) fokussierte Reviews bevorzugen
- Review mit Kontext (Anforderungen, Ziel) durchführen
- eigene Annahmen hinterfragen
- auf Verständlichkeit und Stil achten (Semantik von Bezeichnern etc.)

für Autoren

- Reviews nicht persönlich nehmen
- Nachfragen ggf. als Hinweis verstehen für Dokumentationsmängel

Voraussetzung für Code Reviews

- der Code ist konsistent formatiert (Code Prettifier)
- triviale Fehler wurden bereits behoben (Linting)
- der Code kompiliert
- (Unit-/Integrations-) Tests laufen durch
- Architektur und zentrale Designentscheidungen sind dokumentiert

Achtung!

Es macht nur Sinn Code Reviews für Code durchzuführen, der grundlegenden Qualitätsstandards genügt.

Code Review von CSS

Exemplarische Kriterien

- keine Redundanzen, keine unnötigen Style Definitionen.
- Konsistente Schreibweise von
 - CSS Variablen
 - Klassennamen
 - IDs
- Sind die Namen (CSS Variablen, Klassennamen und IDs) verständlich und sprechend?
(Korrektes Deutsch oder Englisch!)
- Werden CSS Variablen sinnvoll verwendet? (Z.B. um ein leicht anpassbares Design zu ermöglichen?)
- Werden CSS Layer verwendet, um die Styles zu organisieren und die Wartbarkeit zu erhöhen (Stichwort: Modularisierung)?
- Wird CSS Nesting verwendet, um die Struktur der Styles zu verdeutlichen?
- *Einfachheit und Korrektheit von Selektoren*
(Z. B. keine Verwendung von `!important`.)
- Sind die Selektoren effizient?
- Sind komplizierte Selektoren - falls notwendig - dokumentiert/begründet/nachvollziehbar?
- Konsistente Vorgehensweise in Hinblick auf (z. B.) Responsiveness (z. B. konsistente Verwendung von Größen (rem, rem,...) oder das Farbschema.
- Einfachheit des Layouts?
- Gibt es ein klares Vorgehen (Mobile-first or Desktop-first)?
- Werden Tools eingesetzt, um bei der Formatierung bzw. dem Linting zu helfen.
- Wird modernes CSS verwendet?
(Dies gilt immer in Hinblick auf die Zielplattformen/Browserversionen.)

Code Reviews von HTML

Exemplarische Kriterien

- Ist der HTML Code korrekt?

Achtung!

Folgende Aspekte (Auszug) sollten automatisiert geprüft werden:

Sind IDs einmalig, sind alle Element korrekt geschlossen, ist die Baumstruktur korrekt.

- Benennung von Klassen und IDs:

Sind die Namen konsistent vergeben und verständlich und korrekt geschrieben.

- Ist die Struktur einfach nachvollziehbar oder übermäßig komplex?

(Z. B. gibt es eine hohe Anzahl and verschachtelten DIVs oder gibt es eine sinnvolle Strukturierung der Überschriften?).

- Accessibility?

Nutzt der Code nur `` und `<div>`, oder semantisches HTML?

Werden ARIA Tags sinnvoll verwendet?

Haben Bilder ein `alt` Attribut?

- Keine Verwendung von inlines styles?

- Keine Verwendung von inline JavaScript?

- Keine Verwendung von `
` für Formatierungszwecke.

- Konsistente Verwendung von tags?

(Z. B. ist Code immer in `<pre><code>...</code></pre>?)`

- Werden Eingaben in Eingabefelder geprüft?

- Werden `meta` Tags für die Spezifikation des Viewports sowie weiterer Metainformationen verwendet?

- Werden übermäßig große Bilder/Icons vermieden?

- Sind Links inhaltlich nachvollziehbar?

Code Reviews von JavaScript

Exemplarische Kriterien

- Macht der Code was er vorgibt zu tun?
(D. h. sind Variablen, Klassen, Methoden, ... -namen korrekt und sinnvoll?)
- Werden Sonderfälle und Fehlerzustände behandelt?
- Ist die Fehlerbehandlung konsistent und für den Endnutzer nachvollziehbar?
- Werden keine Fehler „verschluckt“?
- Wird null/undefined korrekt behandelt?
- Wird modernes JavaScript verwendet (z. B. Klassen, `const` und `let` anstatt von `var`, Destrukturierung, Spread und Rest Operator etc.)?
- Wird `eval()` nicht verwendet?
- Werden asynchrone Funktionen richtig verwendet?
- Ist der Kontrollfluss verständlich?
(Nachfragen deuten auf Probleme bzgl. der Verständlichkeit hin.)
- Ist der Code modularisiert?
- Wird auf tief verschachtelte Logik verzichtet?
- Werden teure Manipulationen (des DOMs) auf das notwendige Minimum beschränkt?
- Werden Eingaben validiert (auf Client und **Server** Seite)?
- Ist das Logging von Fehlern (Error) sinnvoll und enthält genug Kontextinformationen?
- Gibt es Testfälle? Falls ja, sind diese ausreichend?
(Codeabdeckung ist hier *nur* ein erster Indikator.)
- Wird nur minimaler globaler Zustand verwendet?
- Ist die (Inline-)Dokumentation ausreichend und korrekt.
- Sind die TODOs, FIXMEs verständlich und umsetzbar?
- Bei verteilten Anwendungen: ist die Aufteilung der Logik nachvollziehbar und sinnvoll (zum Beispiel auch aus Sicht von Cheatingmöglichkeiten)