

Kontrast: Wichtig für Personen mit Sehverschärfung oder Alter ü. 50. Kontraststufe 5:7:1 für u.a. 50 (AA), 15:9:1 für u.a. 80 (AAA). Testing u.a. via Chrome Accessibility Audit oder Wave-Plugin.

Auszeichnung von Medien: Bilder sollten immer einen Alt-Text haben. Art von Bild, Kontext, Inhalt und allenfalls Text als Zitat angeben. (*A comic strip of a confused man saying 'OK'*)

Zoombarkeit: Zoom nicht unterbinden (kein user-scalable = 0, etc.). Schrift soll via Browserseinstellungen separat skalierbar sein (Verwendung von em/rem/px statt px).

Animationen: Animationen sollten abstellbar sein, um Ablenkungen (z.B. bei ADHS), Epilepsie und Migräne zu verhindern (@media (prefers-reduced-motion) {}). Animationen von opacity immer in Kombination mit visibility: visible|hidden verwenden.

Bedienbarkeit mit der Tastatur: Alle wichtigen Elemente sind in der richtigen Reihenfolge fokussierbar (Tab). Fokus soll sichtbar sein. Dazu Standard-Controls nutzen und Controls nicht mit CSS unterstützen (kein flex-direction: reverse, etc.).

Screenreader: Für Unterstützung keine Heading-Levels auslassen, semantische Elemente verwenden (anstatt ARIA-Attribute), versteckte Skip-Links ("Zum Hauptinhalt") am Seitenanfang einbauen, Lang-Attribut setzen, Tabellen-Headings für Zeilen/Spalten verwenden, Captions verwenden, Inputs mit Labels versehen.

Custom-Controls: Nicht selber entwickeln. Bestehende Controls mit angepasstem Styling verwenden (Star-Rating aus Radio-Buttons). Drag & Drop vermeiden. Gute Control-Libraries verwenden.

⇒ Automatische Accessibility Tests sind limitiert. Manuelle Tests immer empfehlen.
⇒ Top 5 Probleme: Kontrast, kein Alt-Text, leere Links, keine Labels, leere Buttons.

Wer braucht was?

Sehbehinderte: Kontraste, Zoombarkeit, Trennung von Inhalt und Layout. **Hörbehinderte:** Audiodeskription, visuelles Feedback, einfache Texte. **Blind:** Gute Dokumentenstruktur, Alt-Texte, Tastaturbedienbarkeit. **Motorische Einschränkung:** Grosse Schaltflächen, Tastaturbedienbarkeit, keine unerwünschten automatischen Aktionen. **Lern- & Aufmerksamkeitsstörung:** Einfache Texte, lesbare Schriftarten, keine ablenkenden Elemente.

⇒ Betrifft 21% der Schweizer Bevölkerung.

Security

OWASP Top 10

A01: Broken Access Control **A02:** Cryptographic Failures **A03:** Injection **A04:** Insecure Design **A05:** Security Misconfiguration **A06:** Vulnerable & Outdated Components **A07:** Identification & Authentication Failures **A08:** Software & Data Integrity Failures **A09:** Security Logging & Monitoring Failures **A10:** Server-Side Request Forgery

Fokusthemen

A01.1: Insecure Direct Object References (IDOR)

Beschreibung: Eine Webseite ist verwundbar, wenn Daten ohne Sicherheitsmechanismen über direkte Referenzen erreichbar sind. **Beispiel:** Die Daten eines Nutzers lassen sich über my-site.ch/userID einsehen. Die Berechtigungen des Aufrufers werden aber nicht geprüft. Ein Angreifer kann nun z.B. solange IDs ausprobieren, bis er auf die Seite eines Nutzers kommt.

Lösung: Alle Seiten mit korrekter Berechtigungskontrolle ausstatten. Bei Express.js z.B. via Middlewares, welche die req.params.id mit der aktuellen Nutzer-ID vergleichen.

A01.2: Cross-Site Request Forgery (CSRF)

Beschreibung: Eine Webseite ist verwundbar, wenn sie Formulare zusammen mit Cookies verwendet und die Herkunft des Formulars nicht überprüft. **Beispiel:** Ein Angreifer bringt einen eingeloggt User dazu, auf evil.ch zu gehen. Auf dieser Seite wird (automatisch) ein Formular an my-site.ch/deleteAccount gesendet. Da der Browser die Cookies der realen Webseiten mitsendet, wird die Aktion im Namen des Nutzers ausgeführt.

Lösung: Formulare mit einem CSRF-Token versehen und beim Request überprüfen. Bei Express.js z.B. das csrf Plugin verwenden. Alternativ keine oder SameSite-Cookies verwenden.

A01.3: Replay Attacks

Beschreibung: Eine Webseite ist verwundbar, wenn Aktionen unbeabsichtigt mehrmals ausgeführt werden können. **Beispiel:** Eine Webseite belohnt die Nutzer für jede korrekte Aufgabe mit Punkten. Ein Nutzer kann nun z.B. eine einzige Aufgabe mehrmals absenden, um so mehrere Punkte zu erhalten.

Lösung: Keine "generische" Lösung möglich. In diesem Fall z.B. die Aufgaben abspeichern und nur einmal zählen.

A02: Cryptographic Failures

Beschreibung: Eine Webseite ist verwundbar, wenn sie veraltete oder riskante Kryptoalgorithmen, mangelte Zufallszahlen, hart-kodierte Passwörter oder ähnliches verwendet. **Beispiel:** Eine Webseite sendet die Logindaten eines Benutzers unverschlüsselt und ohne HTTPS an den Server. Ein Angreifer kann nun z.B. mit einer Network Sniffing Software die Logindaten abhören.

Lösung: Verwendung von HTTPS. Keine sensitiven Informationen in der URL codieren. Externe Auth-Services nutzen.

A03.1: Cross Site Scripting (XSS)

Beschreibung: Eine Webseite ist verwundbar, wenn ein Angreifer seinen Schadcode so auf der Seite abspeichern kann, dass dieser im Browser eines Nutzers ausgeführt wird. **Beispiel:** Die Eingaben eines Formulars werden ohne "Escaping" an andere Nutzer ausgeliefert. Ein Angreifer kann nun z.B. eingeben und so beliebigen Code ausführen.

Lösung: Escaping/Encoding verwenden. Eingabebereitungen (Sanitizing) via Libraries (XSS, DOMPurify) einbauen. CSP-Header setzen. HTTPOnly-Cookies setzen.

⇒ Für Encoding in HBS {{content}} anstatt {{{content}}} verwenden.

A03.2 Remote Code Execution / Injection

Beschreibung: Ein Server ist verwundbar, wenn ein Angreifer den Server dazu bringen kann, seinen Schadcode (JavaScript, SQL, etc.) auszuführen. **Beispiel:** Die Eingaben in einem Formular werden mittels eval() vom Server in ein bestimmtes Format konvertiert. Ein Angreifer kann nun z.B. while(true), process.exit() (DoS) oder res.end(fs.readidSync(...).toString()) ausführen.

Lösung: Niemals eval(), sondern parseInt(), oder JSON.parse() verwenden. Eingabebereitungen einbauen. Rechenintensive Tasks auslagern (gegen DDOS-Angriffe). Node.js keine Root-Rechte geben. Globale Scopes und Variablen reduzieren.

⇒ setTimeout(...) und setInterval(...) verhalten sich ähnlich wie eval(...).

A07: Identification & Authentication Failure

Beschreibung: Eine Webseite ist verwundbar, wenn sie u.a. Brute-Force-Angriffe erlaubt, schlechte Passwörter zulässt oder keine Multi-Faktor-Authentisierung verwendet. **Beispiel:** Ein Angreifer bekommt Zugriff auf ein Nutzerkonto, indem er alle möglichen Passwortkombinationen durchprobiert. Da das Passwort "1234" war, ging die Attacke nur wenige Minuten.

Lösung: Korrekte Authentisierung einbauen. Auth-Service nutzen.

Login & Password Handling

Grundsätzlich sollte man, wenn immer möglich, einen externen Auth-Service (z.B. via OAuth) verwenden. Eine Middleware stellt dabei sicher, dass Anfragen auf geschützte Bereiche nur durch autorisierte Benutzer erlaubt sind (z.B. via Token-Validierung).

⇒ Belagten Firmen mit eignen OAuth-Teams sind eigene Auth-Services in Ordnung.

⇒ Möglichkeiten: OpenID Connect, OAuth, Passwordless (Magic-Links, WebAuthN, TOTP).

Weiteres

CSP-Header: Mechanismus, um das Laden von Ressourcen auf die angegebenen Domains zu beschränken (Content-Security-Policy: default-src self trusted-service.com). **HTTPOnly/SameSite-Cookie:** Blockiert den Zugriff von Client-Scripts sowie von anderen Webseiten. **Secure-Cookie:** Begrenzt Cookies auf HTTPS.

Testing

Testarten

Static: Statische Code-Analyse. **Unit:** Testen einer einzelnen Komponente (Klasse, Modul, etc.). **Integration:** Testen von mehreren Komponenten. **E2E/System:** Testen von allen Komponenten. **Funktionen:** Testen von konkreten Anforderungen (Use-Cases). **Regression:** Testen auf potenzielle Fehler nach Code-Änderungen. **Weiteres:** Testen von Security, Usability, Performance, Stress, etc.

⇒ Oft wird dabei von einem System/Under Test (SUT) gesprochen.

Begriffe & Tools

Test-Gruppe/Fixture: Beinhaltet die Tests. **Test-Environment:** Umgebung, in der Tests ausgeführt werden. **Doubles/Mocks:** Fake-Klassen fürs Testen. **Phasen:** Setup, Exercise, Verify, Teardown

Assertion Libraries: Erlauben eine einfache Spezifikation von Tests (Chai, Expect.js). **Test-Runners:** Führen die Tests aus (Mocha, Cypress, Jest). **Mocking Libraries:** Erlauben die Erstellung von Mocks (Proxyquire, Sinon.js). **DOM-Handling:** Erlauben das Testen von Web-UIs (Cypress, Puppeteer).

⇒ Achtung: Teilweise wird auch das Test-Environment als "Fixture" bezeichnet.

⇒ Assertion Libraries könnten auch durch throw new Error (...) ersetzt werden.

Prinzipen & Test Smells

Alles Testen, das kaputt ging oder gehen könnte. Neuer Code ist immer "schuldig". Vor einem Push alle Tests laufen lassen.

Smells: Hard-to-Test Code, Production Bugs, Fragile Tests (Stark abhängig von interner Logik), Erratic Tests (Zufällig Pass/Fail), Developers not Writing Tests, Assertion Roulette (Zu viele Assertions in einem Test), Test Logic in Production Code, Obscure Tests (Zu kompliziert), Slow Tests, Test Code Duplication, Conditional Test Logic.

Mocha/Chai

Erlauben Unit/Integration-Tests. **Syntax:** expect(...).to.equal(val), .to.be.false, .to.be.a(number), .to.deep.equal(array), .to.throw(TypeError), .to.not.be.undefined, .to.have.any.key(key), ...

```
import chai, {expect} from 'chai'
import chaiHttp from 'chai-http'
import jsdom from 'jsdom'
import {app} from './index'
import {use} from 'chai'
import {MochaChaiFixture} from 'mocha-chai-fixture'

const {describe, beforeEach, it, expect} = chai
const {beforeEach, it, expect} = chaiHttp
const {JSDOM} = jsdom

describe('Mocha-Chai-Fixture', () => {
  let aNumber
  beforeEach(() => { aNumber = 1 }) // Setup
  it('Unit: Changed value is new value', () => { // Exercise
    aNumber = 2
    expect(aNumber).to.equal(2) // Verify
  })
  it('Integration: Title contains id of book', () => {
    chai.request(app).get('/books/42').end((err, res) => {
      const dom = new jsdom.JSDOM(res.text)
      const h1 = dom.window.document.querySelector('h1')
      expect(h1).to.contain('42')
    })
  })
  afterEach(() => { aNumber = 0 }) // Teardown
})
```

Cypress

Erlaubt komplexe E2E mit einem Mocha/Chai-ähnlichen Syntax. Läuft in einem eigenen Test-Runner mit Browser-Dashboard.

```
describe('Cypress-Fixture', () => {
  it('E2E: clear button clears the form', () => {
    cy.visit('localhost:3001/orders')
    const input = cy.get('input[name=email]')
    input.type('Typing...')
    .should('have.value', 'Typing...')
    cy.get('#clear-button').click()
    input.should('have.value', '')
  })
})
```

⇒ cy.url() gibt die aktuelle URL zurück. cy.submit() sendet ein Formular ab.

⇒ cy.contains('Hello') gibt das erste Element mit dem Text 'Hello' zurück.

Internationalisierung

Begriffe

Internationalisierung (i18N): Programmieren auf eine Art, sodass Lokalisierung möglich wird. **Lokalisierung (l10N) / Globalisierung (G11N):** Anpassung des Programms (Sprache, Farbe, etc.) an die Sprachregion. **Übersetzung (T9N):** Übersetzung von Texten und Wörtern. **Locale:** Bezeichnung der Sprachregion (z.B. als String).

⇒ Herausforderungen: Sprache, Wörtlinge, Schreibrichtung, Farbbedeutung, etc.

Best Practices

Layout: Elementpositionen an kulturelle Benutzergruppe anpassen (z.B. Leserichtung). Textlänge soll Hierarchie nicht beeinflussen (ungünstige Umbrüche). Layout nicht von der Wortsortierung abhängig machen. **Eingabefelder:** Unterschiede bei Postleitzahl, Telefonnummer, Provinzbezeichnung, etc. beachten. Vollständiger Name anstatt Vor- und Nachname verwenden. Unterschiedliche Datums- und Zahlenformate zulassen.

⇒ Je nach Region/Zielgruppe ist der Lokalisierungsaufwand grösser oder kleiner.

⇒ Nicht alles lässt sich automatisch lokalisieren (Denke: Farbe, Etiquette, etc.).

Umsetzung

Anführungszeichen: Werden bei <...> automatisch basierend auf lang-Attribut gesetzt. **Standard (JavaScript):** ES2021 Internationalisierung. **Bibliotheken:** FormatJS, Polyglot.js, i18next.

```
const regions = new Intl.DisplayNames(['en'], { type: 'region' })
regions.of('CH') // Switzerland

const collator = new Intl.Collator('de')
collator.compare('Ah!', 'Zzz') // Smaller (-1)

const dateFormatter = new Intl.DateTimeFormat(['en-US'])
dateFormatter.format(Date.now()) // 7/9/2023
new Date(Date.now()).toLocaleString(['en-US']) // Alternative

const formatter = new Intl.RelativeTimeFormat(['en'])
formatter.format(-1, 'week') // 1 week ago

const formatter = new Intl.NumberFormat(['de'])
formatter.format(1259.59) // 1.259,5

const formatter = new Intl.ListFormat(['en'])
formatter.format(['Eggs', 'Milk', 'Fish']) // Eggs, Milk, and Fish

const plural = new Intl.PluralRules(['en-US'])
plural.select(1) // 'one': e.g. write '1 cat'
plural.select(18) // 'other': e.g. write '18 cats'

const swiss = new Intl.Locale('gsw') // Alternative to Swiss German
// ES2021 internat
```

Animationen mit CSS

CSS-Transitions

Erzeugen einen Übergang von Zustand A zu Zustand B. Die Frames dazwischen werden automatisch berechnet (Tweening).

transition-property: CSS-Property zum Animieren. **transition-duration:** Animationsdauer. **transition-timing-function:** Beschleunigungsfunktion. **transition-delay:** Zeitdauer, bevor die Animation startet. **Kurzschreibweise** mit transition.

linear ease ease-in ease-out ease-in-out

```
h1 { transition: all 300ms ease-in 0s,
  color 1s linear 0.5s; }
h1:hover { background: red; /* See 'all' */
  color: white; /* See 'color' */ }
```

⇒ Alle Animationen werden über eine Cubic Bezier Curve definiert.

CSS-Transforms

Verändern die Form/Position eines Elements bei der Anzeige. Kann mit transition animiert werden. Rotationen sind im Uhrzeigersinn. Der Ursprung kann mit transform-origin: x y festgelegt werden. Bei 3D-Transformationen kann perspective:[number] auf dem Parent gesetzt werden (höhere Werte wirken "extremere").

translate() scale() rotate() skew()

```
p { transform: rotate(30deg) // rotateX, Y, Z, 3d //
  translate(20px, 5%) // translateX, Y, Z, 3d //
  scale(1.5) // scaleX, Y, Z, 3d //
  skew(15deg); // skewX, Y //
  transform-origin: left top; // Or px, %, ... //
  transition: transform 300ms ease; }

/* Loses translate and scale! */
p:hover { transform: rotate(60deg) skew(30deg); }
```

⇒ Bei mehreren Transforms ist die Anordnung relevant.

⇒ Achtung: Bei Zustandswechseln immer alle gesetzten Transforms wiederholen.

Keyframe Animationen

Erlauben komplexere Animationen mit mehreren Zuständen.

```
@keyframes my-animation {
  0% { background: red; }
  50% { background: green; }
  100% { background: blue; } }

h1 { animation-name: my-animation;
  animation-duration: 5s;
  animation-timing-function: linear;
  animation-iteration-count: 3; /* Or infinite */
  animation-direction: normal; /* Or revert, alternate */ }
```

Die meisten Properties mit quantitativen Werten/Farben lassen sich animieren. Nicht animierbar sind display, auto, url(...), etc.

⇒ Mit der Houdini-API (@propert y) lassen sich auch solche Properties animieren.

⇒ JS-Animationen sind grundsätzlich richtiger, aber nicht so performant.

⇒ SVG-Animationen sind sehr praktisch, wenn der Browser-Support vorhanden ist.

CSS-Präprozessoren

Was sind CSS-Präprozessoren?

Wandeln CSS-ähnlichen Code in CSS um (Sass, PostCSS, Less, etc.).

Sind nicht an CSS-Limitationen gebunden. Erlauben Modularisierung, Wiederverwendbarkeit und wartbaren Code.

Syntactically Awesome Style Sheets (SASS)

Erlaubt Sass- und SCSS-Syntax. Sass: "Einnick"-Syntax mit weniger Schreibaufwand. SCSS: Angepasster CSS-Syntax.

⇒ SCSS-Syntax wird bevorzugt. Sass-Syntax ist historisch bedingt.

Anwendung

Features: Variablen, Verschachtelung, Partials, Mixins, Vererbung, Operatoren, Funktionen, Maps. **Weiteres:** Mixins und Extends funktionieren ähnlich. Mixins generieren mehr Code-Redundanz, erlauben aber Input-Parameter. Extends sind sinnvoll bei thematischen Abhängigkeiten. Im Zweifelsfall Mixins verwenden.

⇒ Beide haben Vor- und Nachteile. Irrer den CSS-Code im Auge behalten.

⇒ Hinweise: Max 3 Ebenen verschachteln. (use verhindert Überschreibungen).

```
@use "colors"; // Partials (use)
$map: $margins; // Partials (import)

@mixin toPosition($x, $y: 15px) { // Mixins with Parameters
  position: absolute; left: $x; bottom: $y; }

@mixin onMobile() { // Mixins
  @content; // Content of importer
  &.narrow { display: block; } // Reference to importer (&) }

ul {
  color: colors.$main-color; // With Namespace (use)
  li { // Nesting
    margin: $main-margin; // No Namespace (import)
  } &:hover { // Parent-Selector (&)
    margin: 2 * $main-margin; // Calculations
  }
}

> a { color: blue; } // With Combinators
div & { // Also Possible (&)
  @include toPosition(10px); // Mixins
  @include onMobile() {
    .wide { display: none; }
  }
}

%abstract-rule { margin: 0; } // Placeholder (not in CSS)
@extend ul; // Inherits all ul-Rules
@extend %abstract-rule; // Placeholder-Inheritance

stylesheet.scss
colors.scss
margins.scss

$main-color: black; // Variables
colors.scss

$main-margin: 10px; // Variables
margins.scss

ul, ol { color: black; }
ul li, ol li { margin: 10px; }
ul li: hover, ol li: hover { margin: 20px; }
ul > a, ol > a { color: blue; }
div ul, div ol { position: absolute; left: 10px; bottom: 15px; }
div ul.wide, div ol.wide { display: none; }
div ul.narrow, div ol.narrow { display: block; }
ol { margin: 0; }

result.css
```

Logik und Rechnen

Einheiten: Numbers (1, 3.5, 5px), Strings ("foo", "bar", px), Colors (blue, #4a3a3f), Booleans, Nulls, Listen, Maps.

Rechnen funktioniert wie in der Physik: 2px+2px=4px, 2*1px=2px, 2px/1px=2, 2px*1px=Error (Kein px), 2+px="2px", "2px"+1px="2px1px", "2px"+"2=Error (String), 2px+1=3px.

```
$list: 2px 4px; // Lists
$map: ("red": red, "blue": blue); // Maps

@function do($val) { // Functions
  @if $val = 2px {
    @return 2 * $val; // 2 * 2px = 4px
  } @else {
    @return $val;
  }
}

@each $elem in $list { // Loops
  body { margin: do($elem); } // Function-Calls
}

// Creates text-red { color: red } and text-blue { color: blue }
@mixin color-modifiers {
  @each $key, $val in $map { // Loops with Maps
    &-#{$key} { color: $val; } // Template Strings
  }
}

text { @include color-modifiers; }
```

Weiteres

PostCSS: Ein CSS-Framework, welches Plugins für das CSS-Parsing und Preprocessing erlaubt (autoprefixer, minifier, etc.). **Built-Tools:** Automatisieren die Erstellung von App-Buildern (WebPack, Vite, Rollup, etc.). Optimieren Performance und Cross-Browser-Support. Beinhalteten u.a. Tree-Shaking, Caching, Resource Loading Optimierung, Polyfills und Transformationen (minify, compress, etc.).

⇒ Bekannte CSS-Frameworks und Component Libraries: Bootstrap, Tailwind, Material UI

⇒ Natives CSS-Nesting soll bald möglich sein (aktuell nur Chrome/Edge).

User-Centered Design

User-Research

1st Rule of Usability: Höre nicht auf deine User, sondern schaue, was sie machen. Befragungen führen zu Spekulation & Wunschkonzent. Kunden dürfen befragt werden (Kunde ≠ Benutzer)

Wichtig: Benutzer, Aufgabe, Tool und Kontext beachten. Du bist nicht der Benutzer. User-Research ist immer möglich. **Problem Space:** Benutzer ist Experte. Kennt seine Aufgaben, Ziele und Probleme. Kann darauf analysiert werden. **Solution Space:** Designer ist Experte. Kann Lösungen für die Probleme entwickeln.

⇒ Kunden und Benutzer müssen unterschiedlich analysiert und befriedigt werden.

⇒ User-Research ist aufwändig, aber unersetzlich für Benutzerorientierte Produkte.

Szenarios

Zeigen eine Geschichte, wie ein Problem aktuell (Problem-Scenario) und in Zukunft (Future-Scenario) gelöst wird. Dokumentieren die Verbesserungen durch das neue Tool. Werden durch Kunden & Benutzer validiert. Als Text oder Storyboard (Sketch, App, etc.) möglich.

Beinhalten: Benutzer (Persona), realistische beobachtbares Problem mit Kontext, Auslöser, Schritte, Lösung oder Fehlschlag.

⇒ Releases sollten sich an den Szenarien orientieren (anstatt den Features).

Befragungen

Grundsätzlich vermeiden. Probleme: Wer wird befragt? Wie wird gefragt (Offen, Suggestiv, etc.)? Was wurde zuvor gefragt?

Qualitative Studien: Wenige Befragungen, die viel aufdecken. **Quantitative Studien:** Viele Befragungen, die wenig validieren.

Navigationsdesign

Mit guter Ausschulderung wissen Nutzer stets: Wo bin ich? Eindeutige Seitentitel, Headers, Icons, etc. und Verwendung von Breadcrumbs. Wo kann ich hin? Eindeutige und sichtbare Navigationselemente. Was ist passiert? Kontinuität und Aktionsfluss durch Animationen, etc. sicherstellen.

Qualität bestimmbar mit **Kofferaum-Test:** Welche Webseite? Welche Seite? Welche Unterseite? Welche Hauptsektionen? Welche Navigationsoptionen? Wo im Gesamtkontext? Wo kann ich suchen?

⇒ mysite.ch / Vision / Our Goal, Inventor, ... / Home, Team, ... / About - Vision / Open Rechts.

⇒ Mit guter Ausschulderung fühlen sich die Benutzer gut aufgehoben (Ziel).

Methoden

Concept Model: Zeigt die Beziehungen zwischen Elementen (Klasse mal Lehrer). **Site Map:** Zeigt die Navigationshierarchie der Elemente (Klasse - Lehrer). **Information Scent:** Links sind so benannt, dass die erreichbaren Ziele erkennbar sind.

Card-Sorting: 1. Zielpunkte (Content Elements) definieren. 2. Gruppen mit "Open Card Sort" bestimmen: 2.5+ Personen der Zielgruppe rekrutieren. 2.6 Zielpunkte in disjunkte Gruppen unterteilen. 2.c Gruppen benennen lassen. 3. Gruppennamen bestimmen (Hypothesen). 4. Gruppen mit "Closed Card Sort" validieren: 4.a 5+ neue Personen rekrutieren. 4.b Gruppennamen vorgeben. 4.c Zielpunkte den Gruppen zuordnen lassen.

Tree-Testing: 1. Aktuelle Navigationsstruktur aufnehmen. 2. Szenarien definieren 3. Auffindbarkeit der Zielpunkte testen.

Usability-Test

Gute Szenarios mit sinnvollen Zielen definieren. Aufs wichtigste fokussieren (schnelle Iteration). Keine konkreten Elemente (Button-Texte, etc.) benennen. Richtige Personen rekrutieren. Vor- und Nach-Interview führen. Personen zum laut denken anregen.

Vorgehen: 1. Benutzergruppe bestimmen. 2. Testpersonen sammeln. 3. Aufgaben (Szenarios) durchspielen. 4. Probleme und Kontext analysieren. 5. Alles dokumentieren.

Usability-Standards

ISO 9241-1: Effektivität: Die Benutzer können ihre Ziele erreichen. Effizienz: Der Aufwand zur Erreichung ist angemessen. Zufriedenheit: Die Benutzer sind gegenüber dem System positiv eingestellt.

ISO 9241-110: Aufgabenangemessenheit, Selbstbeschreibungsfähigkeit, Steuerbarkeit, Erwartungskonformität, Fehleroleranz, Individualisierbarkeit, Lernförderlichkeit

Undo

Gibt Sicherheit, erlaubt Korrektur und vermeidet unnötige Dialoge. Hinweis auf Undo in passive Benachrichtigungen (Snackbar, Toast, etc.) einbauen. Undo immer mit Redo kombinieren.

Design-Erwägungen: Kontext (Alles, ein Feld, ein Upload, etc.). Granularität (Buchstabe, Abschnitt, etc.). Operationen (Ausdrucken/Versenden nicht Undoable).

Weiteres

Wireframes/Mockups: Zeigen einen Sketch des UIs mit möglichst realistischen Inhalt (Paper, Figma, Axure, etc.). Können für Usability-Tests verwendet werden. Vereinfachung: Funktionen können entfernt, versteckt, gruppiert oder verschoben werden.