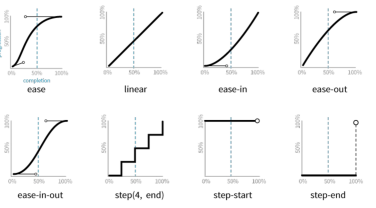
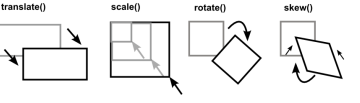


Container/Parent
Aktivierung mit display: grid. Gilt für alle direkten Kinder.
Anzahl und Grösse von Zeilen/Spalten: grid-template-columns grid-template-rows: [measure]. Row-Ausrichtung (pro Zeile): justify-items: start end center stretch. Column-Ausrichtung (pro Zeile): align-items: [justify-items] baseline.
Grid-template-rows: 10px 80% 1fr (3 Zeilen mit definierter Höhe) Berechnung: Grid-Grösse: 100px mit 10px 1fr 2fr = 90px freier Platz ⇒ 90/(1+2) = 30px ⇒ 1fr = 30px, 2fr = 60px;
⇒ Flexbox Eine Primär-Achse, beliebige Anzahl von Elementen, Kinder bestimmen Grösse. Grid: Zwei Primär-Achsen, fixe Anzahl an Elementen, Parent bestimmt Grösse.
Item/Child
Positionenbestimmung: grid-column-start: x ₁ , grid-column-end: x ₂ , grid-row-start: y ₁ , grid-row-end: y ₂ , grid-column: y ₁ /y ₂ , grid-row: x ₁ /x ₂ , grid-area: y ₁ /x ₁ /y ₂ /x ₂ . Row-Ausrichtung: justify-self: start end center stretch. Column-Ausrichtung: align-self: [justify-self]. Row/Column-Ausrichtung: place-self: [justify-self].
Werte
(number)fr: Fraktion des verfügbaren Platzes. Erlaubt Dezimal. Kann nicht schmal als der längste Wort sein (sonst überfließt Wort, wird vermieden). min-content: Breitenanpassung an das längste Wort. max-content: Breitenanpassung an den gesamten Text. min-max (min, max): Gleichmässige Platzverteilung zwischen Min. und Max. fit-content: [length]: Entspricht min(max, [length]). repeat([number])auto-fill auto-fit, [measure]: Wiederholt Wert so oft wie angegeben. auto: Automatische Grössenänderung.
Accessibility
Rechtliches
Seit 2021 gilt in der Schweiz für die öffentliche Verwaltung (Bund, SBB, etc.) der CH-0059 Accessibility Standard (Konformitätsstufe AA). ⇒ Wichtig für Menschen mit Seh-, Hör-, kognitiven oder motorischen Behinderungen.
Themen
Farbenblindheit: Informationen nicht nur mit Farben codieren. Stattdessen Mehrfach-Codierung anwenden (Farbe & Form, Farbe & Icon, etc.). Simulation u.a. via Chrome Dev Tools möglich.
Kontrast: Wichtig für Personen mit Sehschwäche oder Alter ü. 50. Kontraststufe 5.7:1 für u. 50 (AA), 15.9:1 für u. 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. (<i>A comic strip of a confused man saying 'OK'</i>)
Zoombarkeit: Zoom nicht unterbinden (kein user-scalable = 0, etc.). Schrift soll via Browserereinstellungen separat skalierbar sein (Verwendung von em/rem/% statt px).
Animationen: Animationen sollten abstellbar sein, um Ablenkungen (z.B. bei ADHD), 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 umsortieren (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 angepassten Styling verwenden (Star-Rating aus Radio-Buttons). Drag & Drop vermeiden. Gute Control-Libraries verwenden. ⇒ Automatische Accessibility Tests sind limitiert. Manuelle Tests immer empfohlen. ⇒ Top-5 Probleme: Kontrast, kein Alt-Text, leere Links, keine Labels, leere Buttons.
Wer braucht was?
Schbehinderung: Kontraste, Zoombarkeit, Trennung von Inhalt und Layout. Hörbehinderung: Audiodeskription, visuelles Feedback, einfache Texte. Blind: Gute Dokumentenstruktur, Alt-Texte, Tastaturbedienung. Motorische Einschränkung: Grosse Schaltflächen, Tastaturbedienung, keine ungewollten, 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 and Outdated Components A07: Identification and Authentication Failures A08: Software and Data Integrity Failures A09: Security Logging and 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 persönlichen Daten eines Nutzers lassen sich über my-site.ch/user/0 einsehen, ohne dass sichergestellt wird, dass der Aufrufer auch die entsprechenden Rechte hat. Ein Angreifer kann nun z.B. verschiedenen IDs ausprobieren, bis er auf die Seite eines Nutzers kommt.

Lösung: Alle Daten 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 eingeloggten 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 Cookies verwenden (sondern z.B. JWT).
A01.3: Replay Attacks
Beschreibung: Eine Webseite ist verwundbar, wenn Aktionen unbeabsichtigt mehrmals ausgeführt werden können.
Beispiel: Eine Webseite belohnt einen Nutzer für jede korrekte Aufgabe mit Punkten. Der Nutzer kann nun eine einzige Aufgabe mehrmals absenden und erhält so mehrere Punkte.
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 Kryptographien, 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 mit einer Network Sniffing Software die Logindaten abhören.
Lösung: Verwendung von HTTPS. Keine sensiblen 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 in ein Formular werden ohne "Escaping" an andere Nutzer ausgeliefert. Ein Angreifer kann nun z.B. eingeben und so Code auszuführen.
Lösung: "Escaping"/"Encoding" verwenden ({{content}} statt {{{content}}}) in HBS). Eingabebereinigung ("Sanitizing") via Libraries (XSS, DOMPurify) einbauen. CSP-Header setzen. HTTPOnly-Cookies setzen.
A03.2 Remote Code Execution / Injection
Beschreibung: Ein Server ist verwundbar, wenn ein Angreifer den Server dazu bringen kann, seinen Schadcode (z.B. 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.readdirSync(-).toString()) ausführen.
Lösung: Niemals eval(-), sondern parseInt(-) oder JSON.parse(-) verwenden. Eingabebereinigung einbauen. Rechenintensive Tasks auslagern (gegen DoS-Angriffen). 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: Authentisierung korrekt umsetzen oder externen 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). ⇒ Bei grossen Firmen mit eigenen Oauth-Teams sind eigene Auth-Services in Ordnung. ⇒ Möglichkeiten: OpenID Connect, OAuth, Passwordless (Magic-Links, WebAuthN, TOTP, etc.)
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, Secure und SameSite-Cookies: Blockiert den Zugriff von Client-Scripts sowie von anderen Webseiten. (app.use(csurf({cookie: true})) und im Formular <input name=csrf value={{csrfToken}}). Keine Cookies verwenden (stattdessen z.B. JWTs).
Testing
Testarten
Statis: Statistische Code-Analyse. Unit: Testen einer einzelnen Komponente (Klasse, Modul, etc.). Integration: Testen von mehreren Komponenten. E2E/System: Testen von allen Komponenten. Funktional: Testen von konkreten Anforderungen (Use-Cases). Regression: Testen auf systemische Fehler nach Code-Änderungen. Weiteres: Testen von Security, Usability, Performance, Stress, etc.

⇒ Oft wird dabei von einem SUT (System Under Test) gesprochen.
Struktur von Unit-Tests
Test-Gruppe/Fixture: Beinhaltet die Tests. Test-Environment: Umgebung, in welcher die Tests ausgeführt werden. Doubles/Mocks: Fake-Klassen fürs Testen.
Phasen: 1. Setup, 2. Exercise, 3. Verify, 4. Teardown ⇒ Achtung: z.T. wird auch das Test-Environment als «Fixture» bezeichnet.
Tools
Assertion Libraries: Erlauben eine einfache Spezifikation von Tests (Chai, Expect.js). Test-Runner: 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). ⇒ Assertion Library könnte mit throw new Error (...) ersetzt werden (Sinnlos).
Prinzipen
Alles Testen, das kaputt ging oder gehen könnte. Neuer Code ist immer «schuldig», bis das Gegenteil bewiesen wurde. Vor einem Push ins Repo immer alle Tests laufen lassen. Mocks/Doubles mittels Dependency Injection zulassen.
Test Smells
Hard-to-Test Code, Production Bugs, Fragile Tests (Tests sind zu stark von interner Logik abhängig), Erratic Tests (manchmal erfolgreich, manchmal nicht), Developers not Writing Tests, Assertion on Roulette (zu viele Assertions in einem Tests), Test Logic in Production Code, Obscure Tests (zu kompliziert), Slow Tests, Test Code Duplication, Conditional Test Logic (Codeteile werden nicht ausgeführt)
Verwendung (Chai)
Beginnt mit expect(...).to.equal(1).to.be.false, to.not.be.undefined, to.deep.equal(array), to.be.a(number), to.throw(TypeError), to.be.an('array'), that does not include(3), to.have.any.key(key)
import chai, {expect} from 'chai' import chaiHttp from 'chai-http' import jsdom from 'jsdom' import app from './app.js' chai.use(chaiHttp); describe('My Test-Fixture', () => { let number; beforeEach(() => { // Setup number = 1 }) it('Unit: Changed value is now value', () => { // Exercise number = 2 expect(number).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) expect(dom.window.document.querySelector('h1')).to.contain('42') }) }) afterEach(() => { number = 0 }) // Teardown })
Internationalisierung
Begriffe
Internationalisierung (i18N): Programmieren auf eine Art, sodass Lokalisierung möglich wird. Lokalisierung (l10N) / Globalisierung (G1N): 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, Wortlänge, Schreibrichtung, Farbbedeutung, etc.
Best Practices
Layout: Elementpositionen an kulturelle Benutzergruppe anpassen (z.B. Leserichtung). Textlänge soll Hierarchie nicht beeinflussen (ungünstige Umbüche). 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, Etikette, etc.).
Umsetzung
Anführungszeichen: Werden bei <...>/... automatisch basierend auf lang-Attribut gesetzt. Standard: 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('Äpfel', 'Zürcher') // Smaller (-1) const dateFormatter = new Intl.DateTimeFormat(['en-US']) dateFormatter.format(Date.now()) // 7/9/2022 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(1259.56) // 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(10) // 'other': e.g. write '10 cats' const swiss = new Intl.Locale('gsw') // Alternative zum String

Animationen mit CSS
Transitions
Erzeugen einen weichen Übergang zwischen zwei Zuständen. Die Frames dazwischen werden automatisch ausgerechnet (Tweening).
transition-property: CSS-Property zum Animieren. transition-duration: Animationsdauer. transition-timing-function: Beschleunigungsfunktion. transition-delay: Zeitdauer, bevor die Animation startet.
Kurzschreibweise mit transition. Erlaubt auch mehrere Übergänge.

<pre>h1 { transition: all 300ms ease-in 8s, color 1s linear 0.5s; } h1:hover { background: red; /* See 'all' */ color: white; /* See 'color' */ }</pre>
⇒ Alle Animationen werden über eine Cubic Bezier Curve definiert.
Transform
Verändert die Form und Position eines Elements, wenn es angezeigt wird. Aber über Transitions animiert werden. Rotationen sind im Uhrzeigersinn. Der Ursprung kann mittels transform-origin: horizontal vertical; festgelegt werden. Bei 3D-Animationen lässt sich auf dem Parent-Element das perspective-property setzen (je tiefer der Wert, desto extremer die Perspektive).

<pre>p { transform: rotate(30deg) /* rotateX, Y, Z, 3d */ translate(20px, 50%) /* 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; } /* Looses translate and scale */ p:hover { transform: rotate(60deg) skew(6deg); }</pre>
⇒ Beidrehen Transforms ist die Anordnung relevant. ⇒ Achtung: Bei Zustandswechsel immer alle gesetzten Transforms wiederholen.
Keyframe Animationen
Erlaubt die Animation von einer Serie von Zuständen.
<pre>@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 reverse, alternate */ }</pre>
Die meisten Properties mit quantitativen Werten oder Farben lassen sich animieren. Nicht quantifizierbar sind z.B. border-style, display, auto, url(-), etc. ⇒ Mit Houdini (@property) lassen sich neu auch solche Properties animieren.
Weiteres
JS-Animationen: Mächtiger, aber nicht performant (CSS bevorzugt). SVG-Animationen: Sehr praktisch, wenn Browser-Support vorhanden.
CSS-Präprozessoren
Wandeln CSS-ähnlichen Code in CSS um (z.B. Sass, PostCSS, Less). Sind nicht an CSS-Limitationen gebunden. Erlauben Modularisierung, Wiederverwendbarkeit und wartbaren Code.
Syntactically Awesome Style Sheets (SASS)
CSS-Präprozessor. Erlaubt Sass- und SCSS-Syntax. Sass: "Einkürz"-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 Endfunktionen ähnlichen. Mixins generieren mehr Code-Redundanz, erlauben aber Input-Parameter. Extends sind sinnvoll bei thematischen Abhängigkeiten. Im Zweifelsfall Mixins verwenden.

⇒ Beide haben Vor- und Nachteile. Immer den CSS-Code im Auge behalten. ⇒ Maximalbis zu 3 Ebenen verschachteln (Lesbarkeit) und "use" bevorzugen (Überschreibungen).
<pre>{use "colors"; // Partials (use) @import "margins"; // Partials (import) @mixin toPosition(\$x, \$y: 10px) { // 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 { margin: \$main-margin; // Nesting &:hover { // No Namespace (import) margin: 2 * \$main-margin; // Parent-Selector (&) 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) ol { @extend ul; // Inherits all ul-Rules @extend %abstract-rule; // Placeholder-Inheritance } @stylesheets.scss</pre>
<pre>\$main-color: black; // Variables _colors.scss \$main-margin: 10px; // Variables _margins.scss</pre>
Transform
<pre>ul, ol { color: black; } ul li, ol li { margin: 10px; } li:li: hover, ol li:li: hover { margin: 20px; } ul > a, ol > a { color: blue; } div ul, div ol { position: absolute; left: 10px; bottom: 10px; } div ul .wide, div ol .wide { display: none; } div ul .narrow, div ol .narrow { display: block; } ol { margin: 0; } _result.css</pre>
Logik und Rechnen
Einheiten: Numbers (1, 3.5, 5px), Strings ("foo", "bar", "px"), Colors (blue, #00aaff3f), Booleans, Nulls, Listen, Maps.
Rechnen funktioniert wie in der Physik!
<pre>\$list: 2px 4px; // Lists \$map: ('red': red, 'blue': blue); // Maps @function do(\$val) { // Functions @if \$val == 2px { @return 2 * \$val; } @else { @return \$val; } } @each \$elem in \$list { // Loops body { margin: do(\$elem); } // Function-Calls } // Creates text-red {...} and text-blue {...} @mixin color-modifiers { @each \$key, \$val in \$map { // Loops with Maps &-#{\$key} { color: \$val; } // Template Strings } } _calculations.scss</pre>
Weiteres
PostCSS: Framework für CSS. Erlaubt Plugins fürs CSS-Parsing/Preprocessing (z.B. autoprefixer, minifier, etc.). Built-Tools: Automatisieren die Erstellung von App-Bundles (WebPack, Vite, Rollup, etc.). Optimieren Performance und Cross-Browser-Support. Beinhalteten u.a. Tree-Shaking, Caching, Resource Loading Optimization, Polyfills und Transformationen (minify, compress, etc.). ⇒ Bekannte CSS-Frameworks: 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 schau, 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, Probleme. Kann darauf analysiert und interviewt 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-Szenario) und in Zukunft (Future-Szenario) gelöst wird. Dokumentieren die Verbesserungen durch das neue Tool. Werden durch die Kunden und Benutzer validiert. Als Text oder Storyboard (Sketch, App, etc.) möglich.

Beinhalten: Benutzer (Persona), realistisches/beobachtetes Problem mit Kontext, Auslöser, Schritte, Lösung oder Fehlschlag. ⇒ Releases sollen sich an den Szenarien orientieren (anstatt des Features).
Befragungen
Grundsätzlich vermeiden. Probleme: Wer wird befragt (Sampling Bias)? Wie wird gefragt (Offen, Suggestiv, etc.)? Was wurde zuvor gefragt? Qualitativ: Wenige Befragungen, die viel aufdecken. Quantitativ: Viele Befragungen, die wenig validieren.
Navigationsdesign
Mit guter Ausschilderung 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 Kofferraum-Test: Welche Webseite? Welche Unterseite? Welche Hauptsektionen? Welche Navigationsoptionen? Wo im Gesamtkontext? Wo kann ich suchen? ⇒ mysite.ch / Vision / Our Goal, Inventor, ... / Home, Team, ... / About - Vision / Oben Rechts. ⇒ Resultat: Benutzer fühlen sich gut aufgehoben.
Methoden
Concept Model: Zeigt die Beziehungen zwischen Elementen (Klasse hat 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.a 5+ Personen der Zielgruppe rekrutieren. 2.b Zielpunkte in disjunkte Gruppen unterteilen. 2.c Gruppen benennen lassen. 3. Gruppennamen bestimmen (Hypothese). 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. Auf 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: Benutzergruppe bestimmen, Testpersonen sammeln, Aufgaben (Szenarios) durchspielen, Probleme und Kontext analysieren. (Alles Dokumentieren)
Usability-Standards
ISO 9241-11: Effektivität: Die Benutzer können ihre Ziele erreichen. Effizient: Der Aufwand zur Erreichung ist angemessen. Zufriedenheit: Die Benutzer sind gegenüber dem System positiv eingestellt. ISO 9241-110: Aufgabenangemessenheit, Selbstbeschreibungsfähigkeit, Steuerbarkeit, Erwartungskonformität, Fehlertoleranz, 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-Entwägungen: Kontext (Alles, ein Feld, ein Upload, etc.). Granularität (Buchstabe, Abschnitt, etc.). Operationen (Ausdrucken/Versenden nicht Undoable).
Weiteres
Wireframes/Mockups: Zeigen einen Sketch der Benutzeroberfläche mit möglichst realistischem Inhalt (Papier, Figma, Axure, etc.). Kann für Usability-Tests verwendet werden. Vereinfachung: Funktionen können entfernt, versteckt, gruppiert oder verschoben werden. ⇒ Weitere Tools (Ginbal, Flowmapp)