

Reine Formsache



Barrierefreie Formulare mit HTML, CSS & JavaScript

Eintippen, Knopf drücken, fertig: Diskussionen in Foren und Kommentare in Blogs, soziale Interaktion in Communities, das Hochladen eigener Inhalte in Videoportalen, Jobangebote und -gesuche in Stellenbörsen, Fahrplanauskünfte und Tickets bei der Bahn oder Bestellungen und Bezahlung in Online-Shops, ja sogar das Einstellen von Inhalten in Redaktionssystemen und Weblogs – all das funktioniert nur mit Formularen. In Zeiten des Mitmach-Webs werden Formulare immer wichtiger. Formulare sind die technische Basis interaktiver Angebote und die Grundlage der Kommunikation zwischen Anbietern und Nutzern einer Webseite.

Gerade bei Formularen gelten die POUR-Prinzipien der WCAG 2.0 in Reinform: Formulare müssen wahrnehmbar (im Englischen: *perceivable*), bedienbar (engl. *operable*), verständlich (*understandable*) und robust (*robust*) sein. In den Richtlinien finden sich nur wenige direkte Vorgaben zu Formularen, aber fast alle Punkte der WCAG sind anwendbar – von HTML über CSS und WAI-ARIA bis zu JavaScript und der serverseitigen Verarbeitung der eingegebenen Daten. Obwohl HTML seit Jahren viele Elemente für barrierefreie und damit bedienerfreundliche Formulare enthält, setzen Webdesigner sie zu wenig ein.

Die technischen Grundfunktionen eines Formulars – erfassen, ausfüllen und abschicken – müssen mit allen möglichen Mitteln der Ein- und Ausgabe gelingen. Konkret heisst das, Anbieter von Webseiten müssen assistierende Techniken wie z.B. Screenreader, Sprachsteuerung oder Spezialtastaturen berücksichtigen. Für öffentliche Anbieter, die den Vorgaben der Barrierefreie Informationstechnik-Verordnung (BITV) unterliegen, ist der Einsatz dieser Mittel klar geregelt: die Verordnung verlangt die Verwendung der korrekten HTML-Elemente und den Einsatz von CSS für die Gestaltung, das gilt auch für Formulare. Alle anderen Anbieter profitieren ebenfalls von einer einfachen und barrierefreien Nutzbarkeit ihrer Formulare.

Die Konzeption der Formulare oder der Anwendung steht vor der technischen Umsetzung. Der Webentwickler Timo Wirth hat die Perspektive der Nutzer in einer Präsentation einmal so formuliert: »»Ein Formular ist kein Datenbankprozess, es ist der Anfang eines Gesprächs««. Damit ein gelungener Dialog entsteht, gibt diese Serie Hinweise zur barrierefreien Umsetzung von Formularen.

Die Serie ist in fünf Teile gegliedert:

Im ersten Teil »Toleranz und Rücksicht« geht es um Grundlagen und Konzeption von Formularen. Der zweite Teil – »Formulardesign: der wichtige erste Eindruck« – dreht sich um die Gestaltung, Nutzerführung und Usability. Das technische Grundgerüst und damit die Markup-Grundlage von Formularen und das Layout per CSS ist Inhalt des dritten Teils »HTML & CSS für Formulare«.

Neben den für die Struktur benötigten HTML-Elementen und CSS für die Gestaltung kommt in vielen Formularen JavaScript für das

Ihre Optionen:

Die Hoffnung aufgeben

Dem Schicksal ergeben

Geschlagen geben

Kapitulieren

Handtuch schmeißen

Flinte ins Korn werfen

Segel streichen

Waffen strecken

Abschied nehmen

Notbremse ziehen

Wenn Sie während der Lektüre das Gefühl haben, dass sich das alles etwas kompliziert anhört, dann können wir Ihnen versichern: Ja, das ist es. Es sollte nicht verschwiegen werden, dass die Erstellung komplexer formular-basierter Anwendungen harte Arbeit ist –

Verhalten zum Einsatz. In Teil 4 geht es um »Dynamik in Formularen – JavaScript & AJAX«.

Im letzten Teil zeigen wir, wie Sie eine erste Qualitätssicherung vornehmen können und geben Tipps für Tests mit echten Nutzern: »Testen von Formularen und web-basierten Anwendungen«.

von der Konzeption über die Gestaltung bis zur technischen Umsetzung. Aber das Ziel professioneller Entwickler sollte immer sein, höchstmögliche Qualität abzuliefern. Wie Wolfgang Beinert im Typolexikon sagte: »»Qualität beginnt mit Q wie Qual««

1. Konzeption: Toleranz und Rücksicht einplanen

Woran erkennt man ein gutes Formular?

Formulare sind so etwas wie Zahnärzte oder Tankstellen: keiner will hin, aber alle müssen irgendwann mal. Aber genauso wie Tankstellen versuchen, die lästige Pflichtübung in ein angenehmes Erlebnis zu verwandeln, so sollten Ihre Formulare alles andere als langweilig sein oder dem Nutzer unnötige Aufgaben aufbürden. Checklisten, wie solche Fehler zu vermeiden sind, gibt es zuhauf im Netz, nur: woran erkennt man ein wirklich gutes Formular?

Jessica Enders formuliert die folgenden vier Kriterien: »Clear, Concise, Clever, Co-operative« – **Klar, Knapp, Klug, Kooperativ** – diese positiven Eigenschaften sind es, die Nutzer von einem Formular erwarten dürfen; dazu kommt noch die gerade für komplexe Abläufe wichtige Eigenschaft der Konsistenz.

– **Klar** bedeutet, dass der Nutzer mit minimalem Aufwand herausfinden kann, was von ihm verlangt wird. Einfluss auf die Klarheit eines Formulars haben z.B. die verwendete Ansprache der Nutzer, das Formular-Layout und die Optionen, die dem Nutzer zur Verfügung stehen. Die Herausforderung für den Webdesigner: die Ziele des Formulars mit den Ansprüchen der Nutzer in Einklang zu bringen. Dafür müssen Sie die Bedürfnisse der Nutzer und den Nutzungskontext kennen.

Die 5 K's guter Formulare:

- **Klar**
- **Knapp**
- **Klug**
- **Kooperativ**
- **Konsistent**

– **Knapp** bedeutet die benötigten Informationen auf eine möglichst effiziente Art zu sammeln. Viele Abfragen (und damit die Anzahl der benötigten Formularfelder) können durch eine genaue Formulierung reduziert werden. Dabei heisst »knapp« nicht zwangsläufig »kurz«. Manchmal kann es sinnvoll sein, zusätzliche Zwischenschritte einzufügen, um auf bestimmte Eingaben des Nutzers zu reagieren und ihm z.B. die Möglichkeit zu geben, Daten zu präzisieren.

Ein Beispiel für ein Formular, das man deutlich knapper gestalten könnte, sehen Sie im folgenden Screenshot: Auch in einer Millionenstadt ist bei den wenigsten Adressen eine solch präzise Abfrage von Stiege, Stock und Tür notwendig, als dass dies grundsätzlich immer und bei allen Nutzern abgefragt werden sollte. Hier hätte man sich besser für ein gemeinsames Feld »Weitere Adresszusätze« oder eine Funktion zur Einblendung der zusätzlichen Optionen entschieden und damit das Formular für die Mehrzahl der

Anwendungsfälle deutlich verschlankt.

– **Klug** bedeutet, dass die kognitive Belastung für den Nutzer reduziert und er nicht vor unlösbare Aufgaben gestellt wird. Die einfachste Art, diese Belastung zu reduzieren ist, die Aufgaben sequentiell und in einer sinnvollen Reihenfolge zu stellen. Online-Formulare haben gegenüber ihren Print-Pendants den enormen Vorteil, dass sie nicht alle möglichen Eventualitäten auf dem gleichen Stück Papier abfangen müssen. Daher sollten intelligente Formulare nur die unbedingt erforderlichen Daten abfragen (viel mehr dürfen Sie aus Gründen des Datenschutzes sowieso nicht) und optionale oder an Bedingungen geknüpfte Eingaben erst dann zeigen, wenn sie benötigt werden. So macht zum Beispiel die Abfrage der Anzahl der schulpflichtigen Kinder keinen Sinn, wenn der Nutzer vorher bei der Anzahl der Kinder eine Null eingetragen hat.

– **Kooperativ** verhalten sich Formulare, die mit dem Nutzer arbeiten, nicht gegen ihn. Kooperative Formulare erfüllen die in sie gesetzten Erwartungen, sie entsprechen dem mentalen Modell, das der Nutzer sich von der Anwendung gebildet hat, sie erklären sich selbst und die einzugebenden Daten, sie geben Hintergrundinformationen und Hilfen. Neben diesen auch auf gedruckte Formulare zutreffenden Kriterien gilt für Online-Formulare, dass sie dem Nutzer vor der endgültigen Datenübergabe die Möglichkeit zur Korrektur geben und dass sie frei von Software-Bugs sind und stabil laufen.

Weder kooperativ noch sonderlich klug erscheint uns das Formular im folgenden Screenshot eines DSL-Verfügbarkeits-Checks: Die Angabe der Hausnummer »13 a« wird vom System mit der Fehlermeldung »Bitte nur Zahlen (0 bis 9) eingeben« quittiert. Dies lässt nur einen Schluss zu: das anbietende Unternehmen ist offensichtlich nicht an Neukunden interessiert, die in Hausnummern größer als 9 wohnen und zu allem Überfluss auch noch einen Buchstaben in der Hausnummer haben.

– **Konsistent** sind Formulare, die einmal gesetzte Regeln konsequent durchhalten und für den Nutzer vorhersehbar sind. Konsistenz bezieht sich dabei auf alle Ebenen des Formulars, d.h. von der Ansprache des Nutzers über die auszuführenden Aktionen bis zum Layout und dem Erscheinungsbild der Kontroll- und Eingabeelemente. Formulare, die die Ansprache des Nutzers wechseln, sind nicht konsistent. Beim folgenden Screenshot eines Anmeldeformulars wird der Nutzer teils in der ersten Person angesprochen (»Andere können mich über meine E-Mail-Adresse finden«, »Mein Konto erstellen«, »Ich möchte Insider-Infos« etc.), dann wechselt der Text an mehreren Stellen in die zweite Person (»Dein öffentliches Profil«, »Wenn Du "Mein Konto erstellen" anklickst« etc.). und das Formular ist nicht vollständig lokalisiert (»Your full name will appear on your public profile«, »Printable version« etc.)

Vollständiger Name

Your full name will appear on your public profile

Benutzername

Dein öffentliches Profil: [http://twitter.com/ BENUTZERNAME](http://twitter.com/BENUTZERNAME)

Passwort

E-Mail

☒ Andere können mich über meine E-Mail-Adresse finden

Note: Email will not be publicly displayed

Menschliche Nutzer agieren jetzt und in Zukunft auf der Basis von Erfahrungen aus der Vergangenheit, seien diese positiv oder negativ. Ein konsistentes Formular sollte sich der erlernten Arbeitsweise des Nutzers bestmöglich anpassen.

Stattdessen treffen Nutzer laufend auf:

- Formulare, die zu lang und zu kompliziert sind, die ein Übermaß an kognitiver Aktivität verlangen und den Fluss unterbrechen;
- Formulare, die nicht klar sind und deren exakter Zweck nicht bestimmt werden kann;
- Formulare, die den Nutzer zwingen, bestimmte Fragen zu beantworten, auch wenn diese für den Nutzer irrelevant sind;
- Formulare, die dem Nutzer die Kontrolle über das Geschehen entziehen und
- Formulare, die den Nutzer »ansprechen« wenn man sie nicht korrekt ausfüllt.

Diese Formulare sind für viele Nutzer lästig aber mehr oder weniger nutzbar. Menschen mit Behinderung treffen auf Probleme, die eine Nutzung unmöglich machen. Eine Bestellung kann nicht durchgeführt werden, wenn der blinde Nutzer den »Bestellen«-Knopf nicht finden kann. Die Beteiligung an einer Diskussion ist für einen sehbehinderten Nutzer nicht möglich, wenn er mit seiner hochgradig auf seine Bedürfnisse angepassten Konfiguration das Kommentar-Formular nicht wahrnehmen kann. Ein Nutzer mit einer motorischen Behinderung wird aufgeben, wenn er mikroskopisch kleine Formularelemente und Icons nicht zielsicher treffen kann und ein Nutzer mit kognitiver Behinderung wird sich nicht durch eine ellenlange, mehrere Seiten dauernde Folge von Formularfeldern kämpfen, deren Sinn sich ihm nicht erschließt; langatmige und im Juristendeutsch verfasste Erklärungstexte und AGBs verschrecken all jene mit einer den Ansprüchen der Site nicht genügenden Schriftsprachkompetenz.

Die Konsequenzen aus nicht barrierefreien Formular-Anwendungen für viele Menschen mit Behinderung sind damit klar: Kein Einkauf oder keine Beteiligung am öffentlichen Leben.

Eine Liste mit typischen Behinderungen und ihren Auswirkungen auf die Web-Nutzung finden Sie beim AEGIS Consortium unter dem Stichwort »Personas« und im auch online verfügbaren Buch »Just Ask« von Shawn Lawton Henry bei »Personas in User-Centered Design«. Die Erkenntnisse aus diesen »Personas« genannten typischen Nutzungsszenarien sollten Sie bereits in die Konzeption Ihrer Anwendung einfließen lassen.

Klug konzipierte Prozesse

Gutes Design ist mehr als nur eine bunte Dekoration und beginnt bereits in der Konzeption. Ein gutes Design nimmt sich selbst zurück und unterstützt den Nutzer in der Bewältigung der Aufgabe. Niemand käme auf die Idee, Formularen einen hohen Spaßfaktor zu bescheinigen, aber gerade weil niemand Formulare ausfüllen mag, ist ein gutes Design der Formulare unumgänglich, wenn sie denn benutzt werden sollen.

Wobei – ein Formular muss ja nicht immer aussehen wie eine Einkommenssteuererklärung oder so beschränkte Funktionen haben wie ein Kontaktformular. Ohne Formulare gäbe es kein Mitmach-Web, denn Angebote wie Twitter, flickr oder Wikipedia sind ohne die Funktionen von Formularelementen gar nicht möglich.

Gerade bei Formularen bestehen eine ganze Reihe Barrieren für Menschen mit Behinderungen: unnötige Eingaben oder die mehrfache Abfrage von bereits vorliegenden Daten sind für Nutzer mit motorischer Behinderung nicht nur lästig, sondern eine echte Hürde. Wenn sich ein Kunde Ihnen gegenüber z.B. schon mit seiner Kundennummer identifiziert hat, dann ist die erneute Abfrage von persönlichen Daten für diese Nutzer sehr zeitraubend; verbunden mit der Gefahr, dass in der Zwischenzeit die Session abläuft und der Benutzer wieder ganz von vorne anfangen muss.

In der Musik-Community last.fm findet sich ein Beispiel: nach Auswahl der Zeitzone (also MESZ oder GMT) stehen in der darauf folgenden Länderauswahl noch Länder und Kontinente, die von dieser Zeitzone aus gesehen auf der anderen Seite des Planeten liegen. Nutzern, die auf die Tastaturbedienung angewiesen sind, aber in einem Land wohnen, dessen Anfangsbuchstabe im letzten Drittel des Alphabets ist, steht nun eine Odyssee durch eine ellenlange Liste bevor. Auch wenn geübte Nutzer kurzerhand den Anfangsbuchstaben drücken: steht Deutschland denn nun unter Germany, Federal Republic of Germany, Bundesrepublik Deutschland oder Deutschland?

Für sehbehinderte Nutzer ist die Orientierung in komplexen Formularen besonders wichtig. Diese Nutzer haben oft sehr individuelle Einstellungen. Die Erfahrung aus den BIENE-Tests der vergangenen Jahre hat gezeigt, dass von der Standardkonfiguration abweichende Einstellungen wie veränderte Schriftgröße, eigene Farbeinstellungen etc. in vielen Designs nicht berücksichtigt werden.

Tipp: wenn sich Ihr Formular hauptsächlich an ein deutschsprachiges Publikum richtet, dann sollten Sie die Optionen eventuell unterteilen in einen ersten Teil mit deutschsprachigen Ländern (D/A/CH/...) und einen zweiten Teil mit dem Rest.

Nutzern mit kognitiven Behinderungen erleichtern Sie das Ausfüllen, indem nicht nur das Formular selbst auf das Nötigste reduziert wird, sondern auch der Rest der Seite, in die das Formular eingebettet ist. Der Nutzer nimmt meist nichts anderes als das Formular auf der Seite wahr – dann können Sie unnötiges Beiwerk auch gleich ganz weglassen und damit eine kognitive Überfrachtung Ihrer Anwendung verhindern. Klar erkennbare Abläufe minimieren den Lernaufwand und helfen allen Nutzern.

Vor den Details der Formulgestaltung stehen einige grundsätzliche Fragen, mit denen Sie bereits in der Konzeptions- und frühen Designphase viele spätere Probleme Ihrer Nutzer verhindern können. Stellen Sie sich folgende Fragen vor der Umsetzung:

- Sind alle Abfragen wirklich nötig? Warum stellen Sie diese Fragen?
- Ist die Abfrage zum jetzigen Zeitpunkt überhaupt angebracht?
- Sind die abgefragten Daten schon vorhanden? Gibt es andere Wege, um an diese Daten zu kommen? Muss der Nutzer hier Ihre Arbeit machen?
- Können Sie optionale Fragen ausblendbar machen? Sind wirklich alle Abfragen zwingend notwendig oder kann man diese auch weglassen? Ein Beispiel: wenn Rechnungs- und Versandadresse identisch sind, dann sollten Sie den Nutzer nicht zwingen, beides auszufüllen, sondern stattdessen eine Option zur Übernahme der Daten anbieten.

Wenn Sie Formulare ausgehend vom Ergebnis gestalten, verhindern Sie fast schon automatisch unnötige Abfragen. Zunächst analysieren Sie das tatsächliche Ziel des Formulars, dann werden die zu Erreichung dieses Ziels unbedingt notwendigen Daten festgelegt, die zur Verarbeitung der Eingaben zwingend erforderlich sind (also üblicherweise der letzte Schritt in einem Formularprozess). Von diesen Daten ausgehend werden nun die Bestandteile des Formulars quasi »von unten nach oben« festgelegt. Das Ergebnis dieses umgekehrten Ansatzes ist meist ein wesentlich schlankeres Formular als bei der üblichen Methode, zunächst einmal alles in eine Anwendung hineinzustopfen, was im schlimmsten Fall ein Komitee oder Formular-Arbeitskreis als unverzichtbare Features festgelegt hat.

Bei dieser Analyse sollten Sie beachten, dass Prozesse in Formular-Anwendungen aus mehreren, in der Regel vier Ebenen aufgebaut sind:

1. Die Anordnung der Abfragen im Formular (das Layout, Hierarchien und Abstände, die Typografie, aber auch sekundäre Elemente wie Fortschrittsanzeigen und Hilfen)
2. Die Fragen und die dazu gehörenden Antworten in ihren verschiedenen Formaten (die kleinsten Einheiten eines Formulars, mit denen einzelne Daten eingesammelt werden)
3. Die Zusammenhänge zwischen den einzelnen Abfragen im Formular (inkl. eventueller Verzweigungen, die üblicherweise in einem Diagramm dargestellt werden)
4. Die Bearbeitung sowie weitere Aktivitäten und Inhalte um das Formular herum (der Workflow selbst, aber auch die Pfade über die Nutzer zur Formularanwendung gelangen).

Dabei bietet sich an, das Design (Design wird von uns im allumfassenden Sinne verwendet, nicht als pure Dekoration der Oberfläche) beim letzten Punkt zu beginnen, weil dieser Punkt die folgenden beeinflussen wird. Bereits zu diesem Zeitpunkt sollten möglichst alle Beteiligten (die Nutzer der Formulare und die Nutzer der eingegebenen Daten im Unternehmen) in diesen Prozess einbezogen werden – nur so können Sie frühzeitig herausfinden, ob die unterschiedlichen Ansprüche an das Formular miteinander vereinbar sind und eventuelle Differenzen bereits auflösen, bevor es beim fertigen Produkt zu spät ist.

Erst nachdem Sie den minimal erforderlichen Satz an Abfragen festgelegt haben sollten Sie sich an die weiteren Punkte machen, denn Änderungen bei den grundlegenden Dingen verursachen unnötige Doppel-Arbeiten, wenn gleichzeitig bereits am Layout einer Anwendung gearbeitet wird, die noch nicht fertig konzipiert ist.

Bei der Entwicklung von web-basierten Formularen werden allzu oft die Vorgaben aus einem Pflichtenheft oder sogar vorhandene Papierformulare 1:1 in Formularfelder und ihre Beschriftungen übersetzt. Dabei sollte man sich zunächst einmal die Abfolge der Abfragen intensiv anschauen. Dann legen Sie fest, welche Abfragen bzw. Antworten jetzt nötig sind oder auf später verschoben werden können. Jede Abfrage muss der Nutzer aufnehmen und verarbeiten, er muss eine passende Antwort formulieren und diese in das

entsprechende Feld eingeben. Jedes Feld das Sie einsparen, erspart ihren Nutzern Arbeit. Das Ergebnis: Ihr Formular kann schneller bearbeitet werden und die Wahrscheinlichkeit wächst, dass der Formularprozess vollständig durchgeführt wird.

Nutzer brechen eine Transaktion bei missverständlich formulierten Abfragen oft ab oder wenn sie sich fragen »Warum wollen die denn ausgerechnet das jetzt von mir wissen?« Die Reise-Site Expedia hat durch Beobachtung von Benutzern festgestellt, dass viele Nutzer ein (zudem noch überflüssiges) Eingabefeld falsch interpretierten, was in Folge zu einer Fehlermeldung des Systems führte. Nachdem das missverständliche Feld kurzerhand entfernt wurde, konnte ein Umsatz-Plus von 12 Millionen US\$ gemessen werden.

Weitere entscheidende Faktoren für den Erfolg eines Formulars sind die Abfolge der Schritte und die Anordnung der Abfragen zueinander. Eine Reihe von Abfragen kann oft besser in Kategorien unterteilt werden kann, die dann getrennt voneinander präsentiert werden. Ebenso wichtig wie die Abfolge ist, dass dem Nutzer, wie Luke Wroblewski es nennt, ein »clear path to completion« gezeigt wird, also ein deutlicher Pfad zur Vervollständigung der nötigen Eingaben: Von den Eingabefeldern und Kontrollelementen bis hin zum Element, dass die Eingaben abschickt, muss für den Nutzer klar ersichtlich sein, was von ihm verlangt wird und was der nächste Schritt ist. Der Nutzer muss möglichst effizient und zufriedenstellend das Ziel erreichen, dass er mit dem Ausfüllen des Formulars verfolgt: ein getätigter Einkauf, eine erfolgreiche Registrierung oder das Einstellen von Inhalten. Formulare, die Eingaben lose auf einer Seite verteilen und keine logischen Abläufe erkennen lassen, beeinträchtigen die Fähigkeit des Nutzers, passende Antworten auf die Fragen zu finden und den Prozess zu absolvieren.

Zur umfassenden Information des Nutzers gehört eine Anzeige, wie viele Schritte in einem mehrteiligen Formularprozess bereits bewältigt sind und wie viele dem Nutzer noch bevorstehen. Hier ein Screenshot einer solchen Fortschrittsanzeige aus der Anmeldung zum BIENE-Wettbewerb 2010:



Ein indirekter Nebeneffekt der Reduzierung auf das Nötigste ist der, dass Ihre Anwendung dadurch automatisch performanter wird. Große Websites wie Google oder Amazon beobachten sehr genau die Zeiten, die ihre Websites zum Laden benötigen. Sie stellen dabei immer wieder fest, dass bereits kleine Verzögerungen von wenigen hundert Millisekunden teilweise negative Auswirkungen auf den Traffic im zweistelligen Bereich haben. So sprach Marissa Mayer, Vice President von Google, auf der Web2.0-Konferenz im Jahre 2006 davon, dass nur eine halbe Sekunde Verzögerung im Aufbau der Seiten einen 20%igen Einbruch im Traffic nach sich zogen.

Notwendige Komplexität

Daher ist der gegenwärtige Trend zur Reduzierung und Vereinfachung gerade aus Sicht der von Barrieren Betroffenen zu begrüßen – ehrlicherweise allerdings nur bis zu einem gewissen Punkt. Das Leben ist komplex, und unsere Werkzeuge müssen in gewisser Weise diese Komplexität abbilden können. Auch scheinbar simple Dinge können verwirrend sein, genauso wie es komplexe Dinge gibt, die sehr gut verständlich sind. Es geht also nicht darum, Features durch Wegnehmen zu reduzieren, sondern die notwendigen Features so zu gestalten, dass sie von den Anwendern genutzt werden können – erst das ist gutes Design, und das Design von Einfachheit gehört bekanntlich zu den schwierigsten Aufgaben dieser Disziplin.

In der Summe lässt sich die Komplexität einer Formular-Anwendung nicht reduzieren, Sie können nur die Komplexität auf andere Schultern verlagern, z.B. vom Frontend, das der Benutzer sieht zum Backend, an dem der Administrator sitzt.

Unnötige Komplexität

Gerade bei Formularen oder web-basierten Anwendungen gilt das alte Netz-Mantra, dass man bei dem, was man ausliefert, strikt sein sollte und bei dem, was man akzeptiert, tolerant sein sollte. Diese natürliche Fehlertoleranz gegenüber dem Nutzer nimmt einer Anwendung schon viel an möglicher Komplexität, ist aber leider viel zu selten zu beobachten. Dabei würden einige kurze Tests mit echten Nutzern aufzeigen, was diese falsch machen können (*und werden!*) und damit Hinweise geben, wie man diese Fehler auf Seiten des Anbieters am besten von vornherein verhindert.

Ein Beispiel für mangelnde Fehlertoleranz ist der Zwang, Eingaben so zu tätigen, wie es der Anbieter gerne hätte. Die dahinter stehenden Anforderungen oder Konventionen müssen dem Nutzer jedoch nicht unbedingt bekannt sein. So sollten Sie bei Eingabefeldern für Telefonnummern lediglich ein Feld anbieten und in diesem alle Schreibweisen akzeptieren, statt es auf zwei (Vorwahl/Durchwahl) oder sogar drei (mit Ländervorwahl) aufzuteilen. Unterschiedliche Eingaben (+49 – 22 8 – 20 92 0 oder 022820920) können Sie serverseitig auflösen, zumal viele dieser Schreibweisen durch Normen wie die DIN 5008 sogar standardisiert sind. Eine Untersuchung bei einem australischen Mobilfunk-Anbieter hat ergeben, dass es bis zu 40 Varianten gibt, in denen Nutzer ihre Telefonnummer eingeben, wenn man ihnen die Möglichkeit dazu lässt. Dabei müssen Sie noch nicht mal besondere Hinweise auf die Möglichkeit der freien Eingabe hinterlegen – Nutzer tendieren bei fehlenden Hinweisen eher dazu, Daten in der für Sie gewohnten Form einfach einzugeben.

The screenshot shows a web form for entering a mobile number. At the top, a red-bordered box contains the error message: "Rufnummer enthält ungültige Zeichen". Below this, the form is titled "Mobilfunknummer". A text instruction reads: "Zum sicheren Abschluß Ihrer Adresssicherung erhalten Sie eine HandyTAN per SMS. Bitte halten Sie dafür Ihr Mobiltelefon bereit. ?". There are two input fields: "Ihre Vorwahl*" with a dropdown menu showing "0173", and "Ihre Mobilfunknummer*" with a text box containing "275 0643". The "Ihre Mobilfunknummer*" field is highlighted with a red border, indicating it is the source of the error.

Eine weitere Unterteilung in mehrere Textfelder wäre hier in der Regel nicht barrierefrei zu nutzen, da z.B. ein Label immer nur für ein einziges Formularelement definiert werden kann und somit eine logische Verknüpfung mit weiteren Feldern nicht möglich ist. Besonders für blinde Nutzer ist dies ein Problem – sie werden versuchen, die kompletten Daten im ersten Feld einzugeben und regelmäßig an dieser Stelle scheitern. Bei kalendarischen Daten hat sich mittlerweile durchgesetzt, zusätzlich zu den Eingabe- oder Auswahlfelder für Tag/Monat/Jahr auch noch einen echten Kalender in Form eines so genannten Date-Pickers anzubieten. Dies hat den Vorteil, dass hiermit in einem einzigen Kontrollelement umfassendere tabellarische Daten (Kalendertage, Kalenderwochen, Wochenenden etc.) angezeigt werden können, womit man dem Nutzer eine Menge Denkarbeit abnimmt.

Gleiches gilt für alle anderen Zahlen- oder Textfelder, in denen der Nutzer freie Eingaben machen kann, wie Kreditkartennummern, Datumsangaben etc. Auch hier sollten Sie Eingaben mit und ohne Leerschritte, mit oder ohne Zeichen wie / für die Trennung in Telefonnummern oder den durchaus üblichen Punkt als Trennung für Tausender akzeptieren. Besonders fatal auf die Kundenzufriedenheit dürften sich Abfragen auswirken, bei denen das System selbst Daten in einer bestimmten Schreibweise ausgibt, diesen Vorschlag dann aber wie im folgenden Screenshot als Eingabe nicht mehr akzeptiert.

Zwischensumme	12.698 °P
Versandkosten 	0,00 €
Endsumme	12.698 °P
Es sind lediglich gültige Punktwerte erlaubt. Das System hat Ihre einzulösende Punkte automatisch angepasst.	
Einzulösende Punkte	<input type="text" value="12698"/> °P
Zuzahlung	0,00 €
➤ NEU BERECHNEN	
Sie haben noch 814 °P Mehr Prämien aussuchen	
➤ JETZT BESTELLEN	

Werden Daten zwingend in einem bestimmten Format oder in einer festgelegten Länge benötigt, dann sollten Sie dem Nutzer vor der Eingabe erklären, was zulässige Eingaben sind. Diese Eingaben validieren Sie dann in einem weiteren Schritt. Bei einer natürlichen, in der Art der Sache liegenden Begrenzung der Eingabe (z.B. maximal 160 Zeichen beim SMS-Versand oder 140 Zeichen bei Twitter) können und sollten Sie sogar einen Zähler anbieten, der die verbleibenden Anschläge anzeigt. Generell gilt die Regel bei solchen Begrenzungen: was im `maxLength`-Attribut steht, sollte auch noch mal im Klartext daneben stehen (idealerweise im Label des betreffenden Feldes, wie der folgende Screenshot aus der Einreichung zum BIENE-Wettbewerb zeigt):

Sie haben nicht alle Kriterien der Selbsteinschätzung ausgewählt. Bitte nennen Sie uns kurz die Gründe. Falls Ihnen dazu die 1.000 Zeichen des Formulars nicht ausreichen, können Sie dies auch in der Projektbeschreibung erläutern:

(Maximal 1000 Zeichen; Sie haben noch **996** Zeichen zur Verfügung.)

test

Wenn Sie die Länge von Eingaben begrenzen, dann sollte sich Ihr HTML-Code auch daran halten und nicht stattdessen Fehler provozieren. Beispielsweise Passwort-Felder mit der Beschriftung, dass ein Passwort nur 16 Zeichen lang sein darf, die dann zunächst die Eingabe von mehr als 16 Zeichen zuließen, nur um danach eine entsprechende Fehlermeldung auszugeben.

Um beim obigen Kalender-Beispiel zu bleiben: hier kann es durchaus sinnvoll sein, nicht mehr wählbare (weil z.B. in der Vergangenheit liegende) Daten kurzerhand nicht mehr anzuzeigen oder zumindest optisch und technisch zu deaktivieren. Wenn in einem Produkt-Konfigurator bestimmte Optionen nicht miteinander kombinierbar sind, dann sollten Sie den Nutzer nicht dazu verleiten, diese »Fehl«-Konfiguration zu tätigen, nur um ihm hinterher eine Fehlermeldung zu präsentieren. Was nicht geht hat im Formular oder in der Anwendung nichts zu suchen.

Beim Vergleich verschiedener Systeme, z.B. im e-Commerce-Bereich, stößt man immer wieder auf Umsetzungen, die zwar formal barrierefrei sind und alle technischen Kriterien erfüllen, die aber so komplex strukturiert sind, dass sie kein Nutzer mehr (trotz der technischen Zugänglichkeit) versteht. Einen interessanten Vergleich fanden wir in der Analyse zweier im Wettbewerb zueinander stehender Finanz-Programme. Mark Hedlund, der Entwickler eines Konkurrenz-Produkts schreibt hier:

»I was focused on trying to make the usability of editing data as easy and functional as it could be; Mint was focused on making it so you never had to do that at all. Their approach completely kicked our approach's ass.«

<zit>

Ein weiteres Beispiel für unnötige Komplexität sind die verbreiteten Funktionen zur »erweiterten Suche«, die viele Websites anbieten. Es muss nicht immer ein solcher Extremfall wie in diesem Beispiel sein (Screenshot), und es gibt sicher berechnigte Ausnahmen wie z.B. umfangreiche Bibliotheks-Recherchen, aber im Regelfall gilt für die allermeisten Content-Sites: wenn ein Nutzer die einfache Suche nicht versteht, hilft ihm die erweiterte Suche meist auch nicht weiter.

Bei der Abfrage von Adressen können Sie bereits vorhandene mentale Modelle der Nutzer berücksichtigen und damit mögliche Komplexität entschärfen. Durch jahrelanges Training im praktischen Umgang haben Nutzer eingebrannte Vorstellungen davon, wie eine Adresse aufgebaut ist. Generell gehen Adressen vom Spezifischen (Name zuerst) zum Allgemeinen (Land zuletzt), die internationalen Unterschiede liegen meist nur in der Anordnung der Daten zwischen diesen beiden Polen. Genau diese machen aber einen Adressblock erst auf den ersten Blick erkennbar: wenn die Kombination aus Straße/Hausnummer/Postleitzahl/Ort entsprechend den Sehgewohnheiten präsentiert wird, ist der Adressblock und damit die Art der einzugebenden Daten (zumindest für sehende Nutzer) erkennbar, auch ohne dass sie die Feldbeschriftungen gelesen haben.

Wenn Sie jedoch mehrsprachige Inhalte und Funktionen anbieten, dann sollten Sie Ihren Besuchern mit Adressen in anderen Ländern den gleichen Komfort bieten, sodass auch diese auf einen Blick erkennen können, wo der CAP, Code postal, ZIP Code oder Postcode einzugeben ist. Hierzu müssen Sie unter Umständen Anordnung und Größe der Felder anpassen, dieser einmalige zusätzliche Aufwand wird durch höhere Komplettierungsraten wettgemacht.

Eine unschöne Marotte von Formularen ist die doppelte Abfrage von E-Mail-Adressen, z.B. bei Registrierungen. Das vermeintliche Ziel ist klar: damit soll die Wahrscheinlichkeit einer Fehleingabe halbiert werden, da man ansonsten die Gültigkeit einer E-Mail-Adresse nur durch eine Mail an ebendiese verifizieren kann. Das Problem mit dieser Methode:

Weitere Tips zur Berücksichtigung internationaler Adress-Formate finden Sie im Artikel »International Address Fields in Web Forms«

- es unterstellt dem Nutzer von vornherein, einen Fehler zu machen,
- es zwingt ihn zu mühsamen Mehrfacheingaben,
- es löst das Problem falscher Eingaben nicht, da die meisten Nutzer einfach die einmal eingegebene E-Mail-Adresse aus dem ersten Feld in das zweite Feld kopieren (und damit eventuelle Fehler gleich mit).

Mittlerweile beherrschen alle modernen Browser das Ausfüllen von Formularen mit Daten aus dem lokalen Adressbuch des Rechners, was den Zwang zum doppelten Ausfüllen ad absurdum führt.

Nicht so eilig!

Für Nutzer mit kognitiven Behinderungen stellen Zeitbegrenzungen und ablaufende Sitzungen eine echte Hürde dar. Diese Nutzer haben oftmals nicht die nötige Ausdauer, um komplizierte Abläufe an einem Stück abzuwickeln oder sie schaffen es nicht in der vorgegebenen Zeit. Bei komplexen oder mehrstufigen Formularen sollten den Nutzern die Möglichkeit haben, Prozesse zu unterbrechen, Zwischenstände zu speichern und zu einem späteren Zeitpunkt wieder aufzunehmen. Meist handelt es sich typischerweise um Anwendungen, bei denen der Nutzer bereits ein Nutzerkonto besitzt und somit identifizierbar ist, eine Nachverfolgung ist also kein Problem.

Username:	<input type="text"/>	Minutes to stay logged in:	<input type="text" value="360"/>
Password:	<input type="password"/>	Always stay logged in:	<input checked="" type="checkbox"/> <input type="button" value="Login"/> Forgot password?

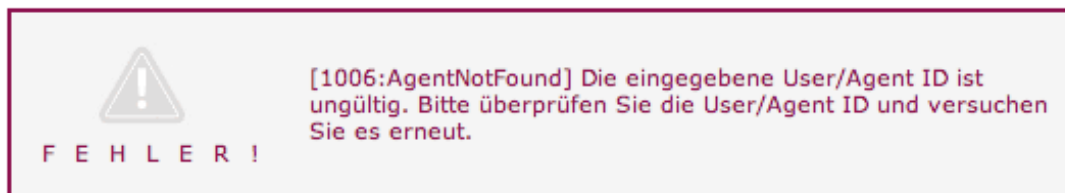
Zum Abschluss eines Formularprozesses muss der Nutzer seine Eingaben kontrollieren können. Insbesondere bei Vorgängen mit erheblicher Tragweite soll erst nach der Kontrolle der Eingaben die eigentliche Übernahme der Formulardaten auf den Server geschehen, um den Vorgang abzuschließen. Bei Änderungen an vorhandenen Datensätzen sollten Sie die nun manipulierte Ursprungsseite nochmals anzeigen, evtl. mit optischen Indikatoren und/oder einer textlichen Zusammenfassung der Änderungen.

Fehlerbehandlung in Formularen

Viele Fehler lassen sich schon von vornherein verhindern, wenn man dem Nutzer die Möglichkeit gibt, die Daten so einzugeben wie er es gewohnt ist. Statt in der client- und serverseitigen Logik der Anwendung auf eine bestimmte Form der Eingaben zu bestehen, werden die Daten dann in einem zweiten Schritt auf dem Server normalisiert (d.h. für die Weiterverarbeitung z.B. einer Bestellung in eine gemeinsame Schreibweise gebracht).

Trotz aller Vorkehrungen lässt sich jedoch nie ganz verhindern, dass Nutzer wirkliche, echte Fehler machen – die Frage ist nur, wie Sie als Anbieter damit umgehen? Sinnvolle und nachvollziehbare Fehlermeldungen können nicht nur Verwirrung bei den Nutzern verhindern, sondern vermeiden auch kostspielige Anfragen beim Kundendienst. Wie Chrissie Brodigan im Artikel »10 Tips on Writing Hero-worthy Error Messages« schreibt: »Error messaging is customer support«.

Lustige Bilder sind kein Ersatz für Fehlerbeschreibungen in Textform, die durchdacht und sinnbildend geschrieben sein müssen, um dem Nutzer wirklich weiter zu helfen. Bei Formularen und insbesondere bei Anleitungen und Fehlermeldungen greifen die Regeln der WCAG bzw. der BITV zum allgemeinen Verständnis und zur einfachen Sprache. So sollten Fehlermeldungen wie die Folgende eigentlich der Vergangenheit angehören:



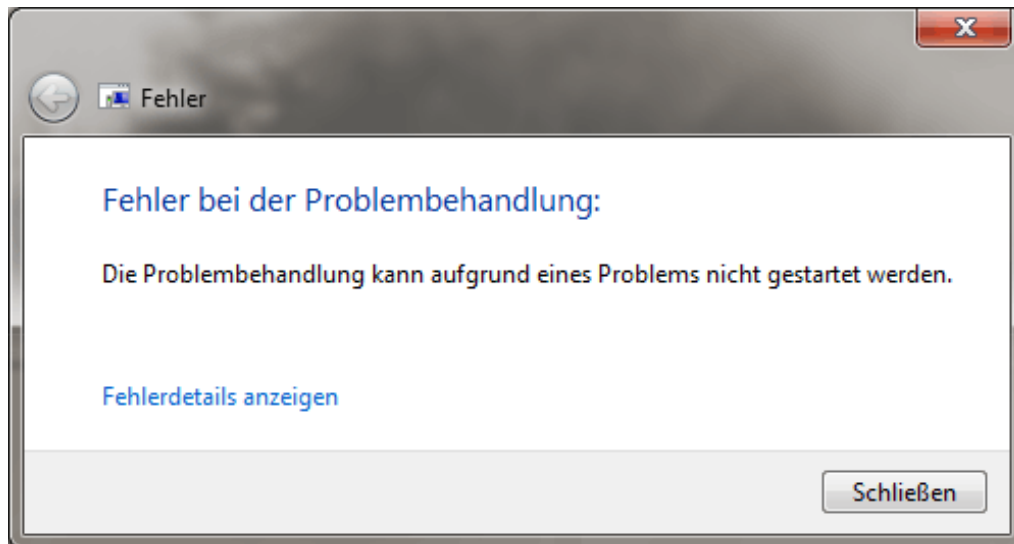
Wir treffen oft auf Situationen, in denen wir ein langes Formular ausgefüllt und abgeschickt haben, nur um anschließend vor einem leeren Formular mit lauter rot markierten Feldern zu sitzen – eine Seite die einem förmlich »FAIL!« entgegenschreit! Zu den unverzeihlichsten Fehlern, die eine Anwendung begehen kann gehört, dass alle Daten verloren gehen, wenn der Nutzer auch nur ein Feld falsch bedient hat. Wenn Sie nicht gerade in einem Bereich mit extremen Ansprüchen an Datenschutz und -sicherheit operieren, dann sollten einmal korrekt eingegebene Daten (natürlich mit der Ausnahme von Passwörtern) erhalten bleiben, sonst bricht der Nutzer an dieser Stelle die Transaktion frustriert ab.

Die Daten über Stellen im Prozess, an denen Nutzer abbrechen, besitzen Sie eventuell bereits: die Logfiles des Servers, deren Analyse die Schwachpunkte zeigt, an denen Nutzer aufgeben. Untersuchen Sie diese Daten bezüglich der Fragen:

- Welche Felder werden falsch ausgefüllt?
- Was haben Nutzer in diese Felder eingegeben (oder vergessen einzugeben)?
- Was sind die häufigsten Fehlermeldungen?
- Wie viele (Fehl-)Versuche benötigen die Nutzer?
- Wie hoch ist die Abbrecher-Quote (und an welcher Stelle wird abgebrochen)?

Verständliche Hilfen

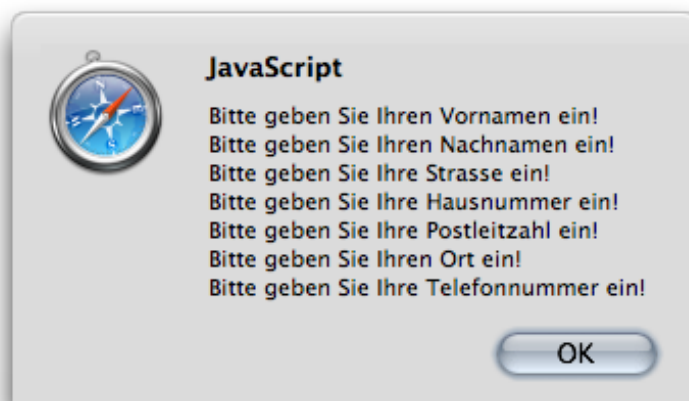
Schon in der Konzeptionsphase sollten Sie sich über die Art der Rückmeldung auf Eingabefehler Gedanken machen und diese auch testen, egal ob Sie mit Wireframes oder Prototypen arbeiten. Diese Überlegungen haben erhebliche Auswirkungen auf die technische Umsetzung. **Dass** Sie Ihre Nutzer über Fehler informieren sollten, versteht sich von selbst. **Wie** Sie Ihre Nutzer über Fehler informieren ist eine Frage der Philosophie und des Kontextes innerhalb der Anwendung. Die einzige Regel, die immer und überall gilt: wenn ein Fehler passiert, dann sollten Sie den Nutzer möglichst schnell und möglichst »laut« benachrichtigen.



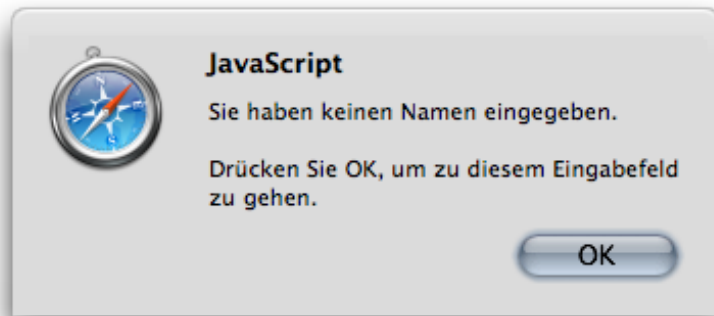
Quelle: laborenz.de/lab-o-log

Wann und in welcher Intensität dies geschieht ist Gegenstand langer Debatten unter Webdesignern und spiegelt die unterschiedlichen Ansätze der verschiedenen Betriebssysteme wieder. Typische Windows-Anwendungen konfrontieren den Nutzer eher mit zu vielen Meldungen (»Es ist nichts passiert. Bitte drücken Sie OK oder Abbrechen, um keine Änderungen zu bestätigen«), Betriebssysteme aus der Unix-Welt wie Linux, BSD oder Mac OS X hingegen sagen nichts, wenn nichts bemerkenswertes passiert ist (geben also in der Regel auch keine Bestätigungen, wenn ein Vorgang ohne Fehler abgelaufen ist).

Die beste Lösung ist der goldene Mittelweg zwischen »zu viel« und »zu wenig« – wo dieser liegt müssen Sie im Kontext Ihrer Anwendung selbst herausfinden und durch Nutzertests verifizieren. Ein typisches Beispiel von »zu viel«, das leider viel zu oft im Netz zu finden ist, sehen Sie im folgenden Screenshot:



Hier kann der Nutzer den Alert nur per OK wegklicken, wird dann aber nicht zu den fehlerhaften Stellen geführt. Wenn Ihr Formular nur aus einer geringen Anzahl Felder besteht und somit die Fehleingaben überschaubar sind, weisen Sie den Nutzer besser einzeln auf die Fehler hin, wie der folgende Screenshot zeigt:



Mit JavaScript (z.B. `document.forms["bestellung"].vorname.select();`) wird dann beim Bestätigen der Warnmeldung per OK-Button der Fokus direkt auf das jeweilige Eingabefeld gesetzt. Diese Methode hat ihre Grenzen: ab einer gewissen Anzahl Felder (und damit potenzieller Fehler) ist es sinnvoller, diese dem Nutzer nicht als endlose Folge von einzelnen Warnmeldungen zu präsentieren, sondern sie als Auflistung im Formular selbst anzuzeigen:

```
<form id="bestellung">
...
<h4>Fehler!</h4>
<p>Folgende Felder müssen ausgefüllt werden, um Ihre Bestellung abzuschließen:</p>
<ol>
  <li><a href="#label-vorname">Bitte geben Sie Ihren Vornamen ein</a></li>
  <li><a href="#label-nachname">Bitte geben Sie Ihren Nachnamen ein</a></li>
  <li><a href="#label-strasse">Bitte geben Sie Ihre Straße ein</a></li>
</ol>
...
<label for="vorname" id="label-vorname">Vorname:</label>
<input type="text" id="vorname">
...
```

Bei einem Reload der Fehlerseite ändern Sie zusätzlich noch den TITLE der Seite sowie deren primäre Überschrift. Hier sollte der klare Hinweis stehen, dass es sich um eine Seite mit Fehlerhinweisen und nicht um die Bestätigungsseite handelt (unter anderem weil der Titel einer Seite im Screenreader immer zuerst vorgelesen wird).

Wenn Sie Hinweise, Hilfen oder Fehlermeldungen ausgeben, sollten Sie dies unter keinen Umständen in Form eines der in letzter Zeit in Mode gekommenen Overlays nach der Art einer Lightbox tun. Diese Art der Präsentation von Dialogen bringt keinen zusätzlichen Nutzen gegenüber echten Dialogen oder Inline-Hinweisen. Im Gegenteil: es treten weitere Probleme auf, man kann mit der auslösenden Seite nicht mehr interagieren, ohne dass die Hinweise verschwinden, denn jeder Klick ausserhalb des Overlays (also auch zur Fehlerkorrektur) schliesst diesen. Dies ist bei mehreren Fehlermeldungen in diesem Overlay besonders schwierig, da der Nutzer gezwungen wird sich sämtliche Meldungen zu merken weil mit dem Schließen die Fehlermeldungen verschwinden.

Wahrnehmbare Hilfen

Auch bei deutlichen Hinweisen auf Pflichtfelder, akzeptierte Daten und Formate, kann und wird es passieren, dass diese Hinweise übersehen werden. Dann muss Ihre clientseitige und die serverseitige Programmierung in der Lage sein, das Formular entsprechend verändert zurückzugeben und nur die fehlenden Eingaben erneut abzufragen.

Dazu gehört auch ein deutlicher Hinweis, welches die fehlerhaften Felder sind. Das kann in Form einer Hervorhebung durch Text, Form und Farbe geschehen. Ein einfaches Umfärben des Textes z.B. in die Signalfarbe rot reicht nicht aus, da 8-10% aller Männer farbfahlsichtig sind und diese Hilfe nicht erkennen.

Verbreitete Screenreader lesen nicht alles vor, was in einem Formular innerhalb des `<form>`-Elements steht. Screenreader verwenden einen sogenannten Formularmodus um die für die Bearbeitung von Formularen benötigten erweiterten Tastaturkürzel zur Verfügung zu stellen. Dieser unterscheidet sich von den herkömmlichen Navigations- oder Vorlesemodi dadurch, dass in Formularen nur die Inhalte der Formularelemente (`<legend>`, `<input>`, `<textarea>` etc.), ihre Beschriftungen (`<label>`) sowie Links (``) vorgelesen werden. Sämtliche anderen Inhalte wie Texte oder Überschriften, die nicht in den klassischen Formularelementen stehen, werden ignoriert. Eine falsche Strukturierung führt dann dazu, dass Hilfen oder konkrete Fragen in einem Formular nicht wahrgenommen werden. So liest ein typischer Screenreader im folgenden Beispiel nur die Label »Ja« und »Nein« vor – die eigentliche Frage bleibt dem Nutzer verborgen:

```
<form>
  <p>Möchten Sie Sardellen auf Ihrer Pizza?</p>
  <input type="radio" name="belag" id="ja" value="anchovis-ja" checked="checked">
  <label for="ja">Ja</label>
  <input type="radio" name="belag" id="nein" value="anchovis-nein">
  <label for="nein">Nein</label>
```

Hier ein separates Fieldset einzuführen und die Frage als Legende (`<legend>Möchten Sie Sardellen auf Ihrer Pizza?</legend>`) zu hinterlegen, würde ein Mehr an Code und damit ein Mehr an Komplexität bedeuten, die an dieser Stelle unnötig ist. Eine mögliche Lösung für dieses »Problem«: wenn Ihr Formular ohne diese Texte Gefahr läuft, nicht verstanden zu werden, dann können Sie die benötigten Texte in Links setzen und diese mit der ID des Labels verknüpfen.

Die Antwort auf die Frage nicht als zwei Radiobuttons, sondern als eine Checkbox und die Frage als deren Label zu hinterlegen ist also die beste Lösung. Eine Checkbox erfüllt hier aufgrund des binären Charakters der Antwort (Haken = Ja, kein Haken = Nein) denselben Zweck wie zwei Radiobuttons.

Barrieren entstehen auch durch die mangelhafte Kennzeichnung der Fehleingaben. Die Hinweise werden meist nicht in unmittelbarem Zusammenhang mit dem betreffenden Kontrollelement gezeigt, sondern weit entfernt davon und nicht sinnvoll verknüpft. Gerade bei nicht-grafischen Zugangsarten wird hierdurch eine Zuordnung schwierig. Oft ist es für den Nutzer einfacher, die gesamte Prozedur von vorne zu beginnen, wenn die Fehlermeldungen nicht sinnvoll mit dem Ort des Fehlers verknüpft sind.

Die oben genannte Technik der lokalen Sprungmarken können Sie sich zu Nutze machen, um zusätzliche Hinweise und Erklärungen (z.B. Hilfen oder Informationen zu Datenschutz und Widerrufsrecht) innerhalb eines Formulars zu verlinken. Im folgenden Screenshot sehen Sie einen Ausschnitt aus dem Anmeldeformular zur BIENE 2010, wo eventuell schwer verständliche Konzepte wie hier die Erklärung, was mit »Transaktionen« gemeint ist, direkt von den entsprechenden Eingabefeldern aus verlinkt sind. Zusätzlich wird der angesprungene Block mit der Erklärung durch die CSS-Pseudoklasse `:target` farblich hervorgehoben – hierzu später mehr im CSS-Kapitel.

Angaben zum Beitrag (Was ist damit gemeint?)

Titel des Beitrags:

URL des Beitrags: *

URL des eingereichten Teilbereichs:

Angaben zum Zweck (Was ist damit gemeint?)

Angebotene Transaktion:

URL dieser Transaktion:

Der Beitrag wird eingereicht in der Kategorie: * (Was ist damit gemeint?)

Transaktionen

Der Online-Kauf eines Buches, die Buchung einer Reise oder das Erstellen eines Gegenstandes im Internet – das sind populäre Beispiele für digitale Transaktionsprozesse. Basis dieser Vorgänge sind komplexe Datenbanksysteme.

Sie ermöglichen es, die persönlichen Daten der Nutzerinnen oder Nutzer zu verarbeiten und beispielweise Kaufvorgänge abzuschließen. Bei den Transaktionsprozessen spielt für den BIENE-Wettbewerb vor allem das Thema Barrierefreiheit eine wichtige Rolle.

Was wäre Webentwicklung ohne die passenden Browser-Bugs? Richtig: Langweilig. Hier gibt es wieder eine

Reihe von Inkonsistenzen und abweichenden Implementierungen in den verschiedenen Browsern. Formularelemente, deren ID von einem Link aus angesprungen wurde, erhalten zwar den :target-Zustand und sind darüber per CSS formatierbar (z.B. mit einer zusätzlichen outline oder border zur besseren Hervorhebung); ob der Fokus aber ebenfalls auf das Formularelement gelegt wird, ist vom verwendeten Browser abhängig. Im Internet Explorer wird der Fokus korrekt auf das angesprungene Formularelement gelegt, dafür versteht dieser in der Version 6 & 7 die :target-Pseudoklasse nicht. Letztere wird von moderneren Browsern zwar verstanden, dafür behalten Safari und Chrome den Fokus auf dem ursprünglichen Link; Firefox hingegen verliert den Fokus komplett, d.h. weder der angeklickte Link noch das angesprungene Formularelement haben den Fokus.

Bedienbare Hilfen

Während die Vorgaben zur Verständlichkeit in Richtlinien wie den WCAG oder Verordnungen wie der BITV uneingeschränkt anwendbar sind, können Sie andere Regeln ruhig etwas liberaler auslegen: so darf man den Sinn des recht pauschalen Verbots von Pop-Ups durchaus hinterfragen, gerade wenn es um Webangebote mit Anwendungscharakter geht. Nach strikter Lesart würden hierunter auch Fehlermeldungen (also echte Alerts) fallen, die jedoch bei korrekter Umsetzung auch in assistierenden Programmen gut nutzbar sind und geeignet sind, den Nutzer vor groben Fehlern zu bewahren.

Ähnliches gilt für Hilfe-Funktionen innerhalb einer Anwendung oder eines komplexen Ablaufs: hier kann es durchaus sinnvoll sein, diese in einem separaten (Pop-Up-) Fenster zu öffnen, damit der Nutzer im Bedarfsfall beides im direkten Zugriff hat. Auch diese stellen keine Barriere dar, sondern helfen dabei, diese zu beseitigen. Ob diese Pop-Ups angekündigt werden müssen ist Gegenstand langer Debatten, in die wir uns nicht einmischen möchten (zumal viele Arten von Pop-Ups aufgrund ihrer Natur gar nicht angekündigt werden können).

Übertreiben Sie es nicht mit den Hilfen

Die Formulierung und der Umfang von Hilfen bedeutet auch immer eine Balance auf dem schmalen Grat zwischen hilfreich oder fördernd und herablassend oder sogar bevormundend – im Englischen wird hierfür das wesentlich besser passende Wort »patronizing« verwendet, für das es leider keine hundertprozentige deutsche Entsprechung gibt. Die Erfahrung aus Nutzertests hat jedoch gezeigt, dass die Wahrscheinlichkeit, dass Nutzer Hilfen wirklich lesen, umgekehrt proportional zum Umfang der Hilfen ist. Zu umfangreiche Unterstützung der Nutzer kann auch sehr schnell negative Konsequenzen auf deren Performance haben.

So berichtete Christof van Nimwegen in seiner Dissertation »The paradox of the guided user: assistance can be counter-effective (2008)« von einem Experiment, bei dem zwei verschiedene Systeme und die Leistung der jeweiligen Nutzer gegeneinander getestet wurden. Das eine System versuchte die Nutzer möglichst weitgehend zu unterstützen, indem möglichst viele triviale, mechanische Bestandteile der Aufgabe vom System übernommen wurden. Nutzer hatten also mehr Zeit, sich mit der eigentlichen Aufgabe zu beschäftigen. Das andere System wiederum stellte dieses Maß an Unterstützung nicht zu Verfügung, zwang also die Nutzer, sich mehr mit den inneren Details der Anwendung zu beschäftigen:

»One group of interfaces is more user-friendly, in that it attempts to aid the user as much as possible in their given task, for example, indicating options available at a given moment of their task (e.g. greying out and disabling unavailable options, or highlighting available moves). This interface tries to offload as much ›trivial‹ mechanical thinking as possible to the machine to give the user more time to think about the problem itself. The other group of interfaces don't provide this level of help and hints, forcing the user to get better acquainted with the mechanics of each problem and put more thought into how they will want to complete their task.

In essence, one set of interfaces externalizes information, whereas the other internalizes it, in relation to the user. [...] the user-friendly interface relieves its users

<zitat>

from having to commit information to memory, and this information is in turn externalized. When the interface isn't helpful, information is internalized by the user, meaning they have to think longer about the problem and learn more fully the inner workings of their task.«

Entgegen alle Regeln und Mantras der Usability stellte sich heraus, dass die zweite Gruppe wesentlich besser bei der Bewältigung der Aufgaben abschnitt, weniger abgelenkt und konzentrierter bei der Arbeit war und zudem das erworbene Wissen auch auf andere Systeme transferieren konnte:

»The findings are very interesting and go against the general trend of simply accepting usability of user interfaces as something completely beneficial to the user. Indeed, the opposite thing happened. People using the more difficult interfaces tended to perform better, were less fazed by distractions and were found more likely to transfer their skills to new interfaces or tasks.

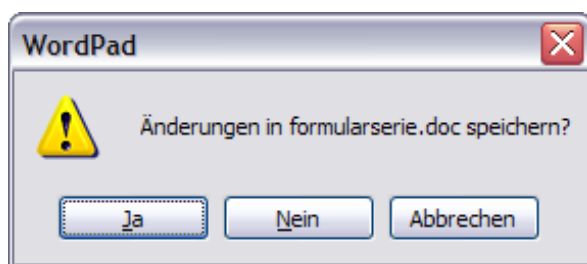
People using the user-friendly interfaces tended to rely on them too much, and so were never fully able to grasp the problem and come up with working strategies for tackling it. Even though the user-friendly interface was meant to relieve them of trivial tasks, they instead relied on the computer as a crutch, stumbling around on their way to an eventual solution with the help of the interface.«

(zitiert nach: »The Dark Side Of Usability«)

Ansprechende Texte

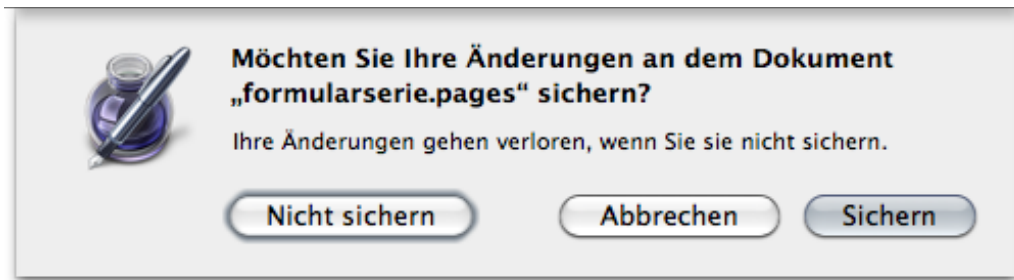
Zum Design der Abfragen, Bestätigungen und Fehlermeldungen gehört auch die Festlegung von Formulierungen, die dem Nutzer den Umgang mit der Anwendung erleichtern. Im Artikel »Usability Tip: Use Verbs as Labels on Buttons « spricht Dmitry Fadeyev davon, Verben mit Aufforderungscharakter statt abstrakter bzw. generischer Begriffe zu verwenden – eine Idee die sich nicht nur auf Dialoge beschränkt, sondern auch auf Feldbeschriftungen, Aufforderungen, Hilfetexte usw. übertragen lässt.

Wir alle kennen die üblichen Buttons in Dialogen: kurze Phrasen die uns auffordern, eine Aktion zu bestätigen oder abzubrechen. Im folgenden Screenshot sehen Sie einen solchen Dialog aus einer Textverarbeitung unter Windows, der angezeigt wird, wenn man ein Dokument schließt ohne es gespeichert zu haben:



Der Dialog fragt, ob man die Änderungen im Dokument speichern möchte. Danach folgen die Buttons »Ja«, »Nein« und »Abbrechen«. Deren Funktion ist aber erst offensichtlich, nachdem man den vorstehenden Text ebenfalls gelesen hat; für sich genommen geht ihr Informationswert gegen Null.

Der nächste Screenshot ist aus einem vergleichbaren Programm unter MacOS X:



Die Nachricht ist ähnlich wie unter Windows, nur geht sie mehr ins Detail und erklärt was passiert, wenn man die Änderungen nicht sichert. Der interessante Unterschied liegt aber in der Betextung der Buttons: hier finden sich nur Verben (»Nicht sichern«, »Abbrechen« und »Sichern«), die alle auch ohne ihren Kontext verständliche Aussagen über die dahinter liegende Aktion machen. Im ersten Beispiel muss man in der Tat die gesamte Dialogbox lesen, um die Aktion zu verstehen und eine Entscheidung zu treffen, im zweiten Beispiel kommt der Nutzer in kürzerer Zeit zu einer klareren Entscheidung.

Eine kurze Checkliste

- Teilen Sie dem Nutzer mit, dass ein Fehler aufgetreten ist.
- Machen Sie deutlich was der Fehler ist und an welcher Stelle er aufgetreten ist.
- Geben sie dem Nutzer die Hilfen die er benötigt, um den Fehler zu beheben oder um einen Ausweg aus diesem Zustand zu finden.

Eine ganze Sammlung hervorragend gemachter Fehlermeldungen finden Sie in der UI Patterns Library unter »Input Feedback«.

Um die Ecke denken

Formulare sind von ihrer Natur her etwas sehr mechanisches – die Menschen, die sie ausfüllen sollen, hingegen eher nicht. Oft ist es für Nutzer einfacher, ihr Anliegen in Textform auszuformulieren, statt Daten aus einer nicht enden wollenden Anzahl von Optionen auszuwählen. Zumal diese Optionen oftmals gar nicht die gesamte Bandbreite der möglichen Eingaben abdecken können (wie oft standen Sie schon vor einem Formular, dass nur ein »Ja« oder ein »Nein« anbot, die aus Ihrer Sicht richtige Antwort wäre aber ein »Vielleicht« gewesen?). Warum also nicht dem Nutzer die Freiheit lassen, die Daten so einzugeben wie er es möchte? Das war die Überlegung hinter der Neugestaltung des Formulars für die Nutzer-Vorschläge beim BIENE-Wettbewerb 2010: dem Nutzer sollte das Formular in Form eines Lückentexts angeboten werden. Auszufüllen waren nur ein paar Lücken in den Sätzen:

»Ich möchte gerne die Webseite: [] für eine BIENE vorschlagen. Mein Name ist: [],
meine E-Mail-Adresse ist: []. An der Seite gefällt mir besonders gut: [].«

Lediglich der URL des Vorschlags (ohne diesen geht es nicht) und eine Begründung, warum man gerade dieses Angebot gut findet, waren Pflichtfelder; der Rest optional.

Vorschlag zur BIENE 2010

Bitte beachten Sie, dass die mit * markierten Felder Pflichtfelder sind, die ausgefüllt werden müssen.

Ich möchte gerne die Webseite *: für eine BIENE vorschlagen.

Mein Name ist: , meine E-Mail-Adresse ist:

An der Seite gefällt mir besonders gut: *

Vorschlag abschicken →

Ob diese Umgestaltung tatsächlich zu der gemessenen Zunahme der Vorschläge geführt hat, können wir nicht wirklich nachweisen (dazu fehlt ein A/B-Test), aber zumindest gab es über dieses Formular 20% mehr Vorschläge als über das herkömmlich aufgebaute Formular des Vorjahres.

Diese Beobachtung deckt sich mit den Erkenntnissen bei anderen Sites, wo vergleichbare Formulare eingesetzt werden: Luke Wroblewski spricht von Zunahmen um 25-40% durch diesen nach einem Kinderspiel benannten »Mad-Libs-Style«. Der Grund scheint zu sein, dass Nutzer durch die Art der Abfrage und der Präsentation des Formulars nicht aus ihrem Aktivität herausgerissen werden; stattdessen reiht sich die Abfrage der Daten nahtlos in die Aktivität ein und unterstützt statt zu unterbrechen.

Dies ist übrigens auch der Grund, warum hier bei Einfach für Alle die Reihung der Formularfelder für Kommentare im Blog von der üblichen Reihenfolge (1. Vorname 2. Nachname 3. E-Mail 4. URL 5. Kommentar) abweicht. Ausgehend von der Überlegung, dass Nutzer ja schließlich primär einen Kommentar und nicht als erstes Ihren Namen abgeben wollen, wird folgerichtig zuerst das Kommentarfeld gezeigt, die sekundären Daten (Name, E-Mail, URL) folgen erst danach.



The image shows a web form for submitting a comment. At the top, there is an orange header bar with a speech bubble icon and the title 'Kommentar abgeben?'. Below the header, the form is set against a light green background. It starts with the label 'Ihr Kommentar:' followed by a large text area with a small asterisk to its right. Below the text area is a tip: 'Tipp: HTML ist nicht zulässig; Webadressen können Sie so: [url=domain.de]Text[/url] eingeben.' This is followed by three input fields: 'Ihr Name:' with an asterisk, 'Ihre E-Mail-Adresse: (wird nicht veröffentlicht)', and 'URL: (wird bei Spamverdacht gelöscht)'. At the bottom is a button labeled 'Kommentar senden'.

2. Formulardesign: der wichtige erste Eindruck

Auch bei Formularen ist der erste Eindruck entscheidend. Der Erfolg eines einzelnen Formulars oder einer ganzen Formular-basierten Anwendung hängt stark von diesem ersten Eindruck ab. Erschlagen Sie Ihre Nutzer mit einer Vielzahl unübersichtlicher Optionen, werden Sie diese Nutzer so schnell nicht wieder sehen. Das Formular sollte intuitiv zu bedienen sein und bei den klassischen Usability-Messgrößen Effektivität, Effizienz und Zufriedenheit gut abschneiden. Dadurch steigt einerseits der Anreiz, das Formular vollständig auszufüllen, auf der anderen Seite wird es weniger Abbrüche und fehlerhafte Eingaben geben.

Die optische Gestaltung

Der erste Eindruck ist vor allem optisch: sind Farbgestaltung und Schrift stimmig, ist das Formular übersichtlich und unterstützt die Gestaltung beim Ausfüllen? Viele praktische Beispiele zeigen, dass auch komplexe Formulare nicht so aussehen müssen wie eine Steuererklärung – lassen Sie sich inspirieren. Je wichtiger das Formular für die Website ist, desto mehr Zeit sollten in seine Optimierung gesteckt werden. Dafür kann auch ein Experte herangezogen werden.

Die allgemeinen Gestaltgesetze wie Prägnanz, Nähe, Ähnlichkeit, Kontinuität, Geschlossenheit gelten auch für das Design von Formularen. Inhaltlich zusammen gehörende Elemente sollten gemeinsam gruppiert werden. Logisch getrennte Einheiten sollten auch optisch getrennt werden.

Stabile Achsen

Visuell werden Formulare durch Achsen gestaltet. Achsen werden durch die Ausrichtung der Beschriftungen und der Eingabefelder zueinander erzeugt. Anwender können diese visuellen Achsen nutzen, um Informationen über die einzelnen Formularbestandteile zu ermitteln:

- Was wird genau verlangt?
- Ist dies ein Pflichtfeld?
- Schliessen sich Optionen gegenseitig aus oder ergänzt ein Feld die Eingaben in einem anderen Feld?
- Wo stehen die Fehlermeldungen?

Achsen entstehen durch Leerräume zwischen oder vor und nach den einzelnen Bestandteilen des Formulars. Diese Leerräume oder whitespaces können und sollten großzügig bemessen sein. Die Achsen werden einheitlich gestaltet, damit kein unruhiger Eindruck entsteht. Beschriftungen und Eingabefelder sollten einheitlich ausgerichtet sein und eine vergleichbare Länge aufweisen.

Im folgenden Screenshot sehen Sie ein Formular mit fünf unterschiedlichen Breiten der Eingabefelder. Da sich Nutzer in unserem Kulturkreis zeilenweise von links oben nach rechts unten durch ein Formular bewegen, entsteht durch die unterschiedlichen Zeilenlängen ein äusserst unruhiger Eindruck.

Rechnungskopf	
Rechnungsnummer *	<input type="text"/>
Rechnungsort *	<input type="text"/>
Rechnungsdatum * (TT.MM.JJJJ)	24.11.2010
Fällig (TT.MM.JJJJ) *	<input type="text"/>
Bestellende Dienststelle *	<input type="text"/>
Steuernummer bzw. UID *	<input type="text"/>
Datum der Bestellung (TT.MM.JJJJ)	<input type="text"/>
Bestell-Nummer	<input type="text"/>
Firmenbuch-Nummer bzw. SV-Nr.	<input type="text"/>
Datum der Leistung (Lieferung) (TT.MM.JJJJ) *	<input type="text"/>
Leistungsempfänger *	<input type="text"/>
Bankleitzahl *	<input type="text"/>
Girokontonummer *	<input type="text"/>
Zahlungsbedingungen	<input type="text"/>

Wohin mit den Beschriftungen?

Die Beschriftung von Eingabefeldern sollte immer mit `label` vorgenommen werden, weil `label` Beschriftung und Eingabefeld eindeutig verknüpft. Schon die Positionierung der Beschriftungen der Kontroll- und Eingabeelemente hat große Auswirkungen auf die Gebrauchstauglichkeit Ihres Angebotes. Ob Beschriftungen nun zeilenweise neben den Formularelementen oder mehrzeilig über den Formularelementen angeordnet werden sollten, ist Gegenstand langer Debatten. Nach Erkenntnissen aus Nutzertests hat sich die Variante der seitlichen Beschriftungen bei längeren Formularen bewährt, während die mehrzeilige Variante eher für kurze, wenig komplexe Formulare geeignet scheint.

Wie die Elemente am besten angeordnet werden sollten, hängt von den jeweiligen Aufgaben des Formulars ab. Es gibt nur wenige allgemeine Empfehlungen.

Wichtige Anhaltspunkte bilden die Human Interface Guidelines der Betriebssysteme Windows, Mac OS und Linux.

Die Betriebssysteme selbst sind nach diesen Leitlinien aufgebaut und die Nutzer sind daran gewöhnt, dass auch Anwendungen im Web sich an diesen Leitlinien orientieren. Bei allen Unterschieden zwischen den einzelnen Leitlinien sind sie sich in folgenden Punkten einig:

- Beschriftungen von Textfeldern und Auswahlfeldern stehen grundsätzlich links neben oder über dem Feld,
- Beschriftungen von Radiobuttons und Checkboxes stehen grundsätzlich rechts neben den Kontrollelementen.

Wenn Sie sich an diese Konventionen halten, helfen Sie den Nutzern beim Ausfüllen des Formulars, weil die Platzierung den Sehgewohnheiten entspricht und so vorhersagbar wird.

Variante 1: Beschriftung seitlich, linksbündig ausgerichtet

Das Diagramm zeigt ein Formular mit der Überschrift 'Überschrift'. Darunter sind vier Beispiele für die Beschriftung von Eingabefeldern dargestellt:

- Label:** Ein Textfeld mit der Beschriftung 'Label' links daneben.
- Langes Label:** Ein Auswahlfeld mit der Beschriftung 'Auswahl' links daneben.
- Extralanges Label:** Ein Textfeld mit der Beschriftung 'Extralanges Label' links daneben.
- Extraüberlanges Label:** Ein Textfeld mit der Beschriftung 'Extraüberlanges Label' links daneben.

- Die Texte der Beschriftungen sind hier gut zu erfassen, weil sie linksbündig ausgerichtet sind. Die Gestaltung wirkt jedoch wegen der zusätzlichen optischen Achse unruhiger. Diese Variante ist am besten geeignet für Formulare mit ungewöhnlichen Eingaben, weil sich die Nutzer stärker mit den Beschriftungen beschäftigen müssen.
- Für dieses Formular-Layout müssen Sie bei der Umsetzung eine sehr stabile float-Umgebung wie YAML, YUI CSS Grids oder ein spezialisiertes Formular-Framework wie Formee einsetzen. Entwickler setzen in diesem Bereich häufig noch auf Layouttabellen, weil viele Browser die nötigen CSS-Eigenschaften nur unzureichend unterstützen
- Ein Vorteil liegt darin, dass diese Variante weniger vertikalen Platz benötigt. Sie kommt damit dem Trend entgegen, dass Monitore immer breiter werden (16:9 statt 4:3).
- Ein Nachteil ist die schlechte Internationalisierbarkeit. Bei längeren Beschriftungen oder vergrößertem Text kommt es schnell zu unerwünschten Umbrüchen wodurch sämtliche Elemente verschoben werden.

Bei kurzen Beschriftungen entstehen große Lücken zwischen Beschriftungen und Eingabefeldern. Vor allem für Sehbehinderte wird es schwierig, zusammengehörende Elemente korrekt zuzuordnen, wie im folgenden Bild zu sehen ist:

A screenshot of a web form titled "Überschrift". It contains seven rows, each with a label "Label" on the left and an empty input field on the right. The labels are short, creating large, empty spaces between them and between the labels and the input fields. This makes it difficult for users, especially those with visual impairments, to associate the labels with the correct input fields.

Im nächsten Bild sehen Sie ein Designbeispiel mit gestreiften Label/Control-Paaren, das diese Assoziation zwar wiederherstellt, durch die enormen horizontalen Abstände jedoch weiterhin sehr ermüdend für das Auge des Nutzers ist:

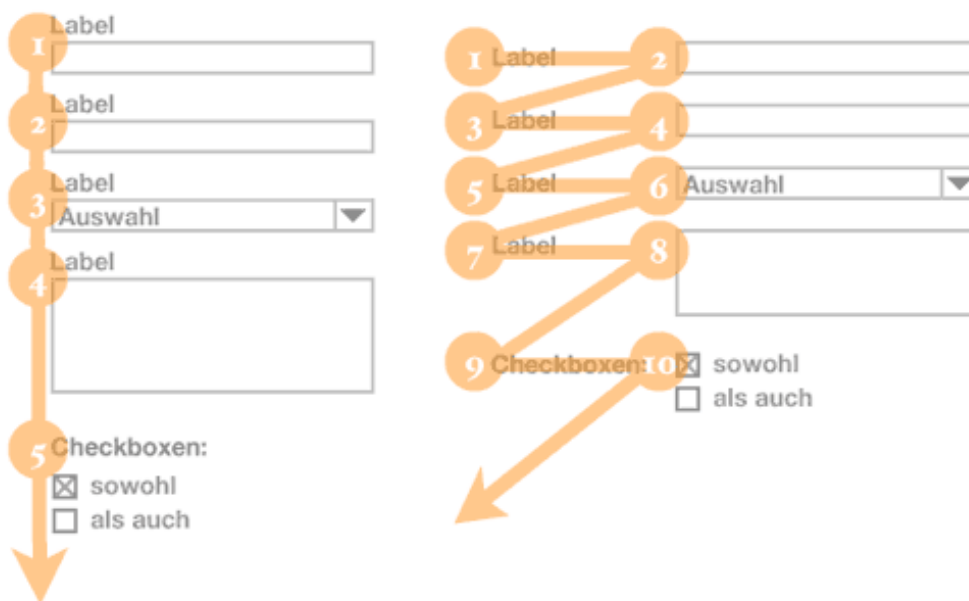
A screenshot of a web form titled "Überschrift". It contains seven rows, each with a label "Label" on the left and an empty input field on the right. The labels and input fields are grouped together by alternating light and dark gray background stripes. While this helps with association, the large horizontal gaps between the groups remain, which is still visually tiring for the user.

Variante 2: Beschriftungen seitlich, rechtsbündig ausgerichtet

- Diese Variante verknüpft Beschriftung und das zugehörige Feld durch die Nähe zueinander. Dadurch kann sich der Nutzer schnell im Formular orientieren. Dies mag einer der Gründe sein, warum die rechtsbündige Ausrichtung in Normen wie der ISO 9241-17 empfohlen wird.
- Diese Variante benötigt weniger vertikalen, dafür mehr horizontalen Platz – problematisch wird dies bei mehrspaltigen Formularen etwa durch `<fieldset>` neben `<fieldset>`.
- Wie in Beispiel 1 beschrieben ist diese Variante schwierig in CSS umzusetzen.
- Durch die Orientierung an der Mittelachse gibt es keinen einheitlichen Zeilenanfang. Dadurch sind Optik und Umbrüche kaum kontrollierbar. Die Beschriftungen sind somit für Nutzer schwieriger zu überfliegen, weil sie links »flattern«.
- Allerdings ist das schnelle Überfliegen zur Orientierung ein nachrangiges Problem bei Formularen (im Gegensatz zu Fließtexten): wir wollen ja gerade, dass sich die Nutzer mit den Feldbeschriftungen beschäftigen und die Eingaben sequentiell in der Reihenfolge ihres Auftretens bearbeiten.
- Problematisch im Sinne der Barrierefreiheit ist diese Variante für stark sehbehinderte Nutzer mit einem brauchbaren Sehrest, die sich mittels Lupenprogrammen visuell orientieren. Bei den teils enormen Vergrößerungsfaktoren kann es dazu kommen, dass der Nutzer unter der Legende (hier: »Überschrift«) lediglich einen großen weißen Bereich wahrnimmt und das erste Formularfeld nicht findet.

Variante 3: Beschriftungen oben, linksbündig ausgerichtet

- Diese Variante verknüpft eindeutig Beschriftung und Eingabefeld. Sie eignet sich nach unserer Erfahrung besonders für sehr einfache Abfragen, wenn die erhobenen Daten dem Nutzer vertraut sind. Die sind zum Beispiel einfache Formulare mit der üblichen Kombination aus Vorname, Nachname, E-Mail, PLZ, Ort, etc. oder Benutzername/Passwort.
- Ausreichende Abstände und Kontraste sind hier besonders wichtig, um ein effizientes Scannen der Beschriftungen zu ermöglichen.
- Diese Variante ist die flexibelste Lösung für Internationalisierungen und variierende Anordnungen, da sie den Labels sehr viel horizontalen Platz lässt und somit auch längere Texte möglich sind. Sie benötigt allerdings mehr vertikalen Platz.
- In vielen Studien (z.B. von Matteo Penzo, der die Effizienz der Anordnung untersucht, oder cpartners »Web forms design guidelines: an eyetracking study«, wo die Zufriedenheit der Anwender untersucht wurde), schneidet diese Version am besten ab, wenn es um den minimalen Zeitaufwand zum Ausfüllen geht. Wie im Artikel »Why Users Fill Out Forms Faster with Top Aligned Labels« schlüssig argumentiert wird, liegt der Vorteil hier an der wesentlich geringeren Anzahl an Punkten, die das menschliche Auge anzuspringen hat, da die visuelle Orientierung entlang des natürlichen Leseflusses von oben nach unten entlang einer einzigen Achse verläuft.



- Einziger Nachteil: das Formular benötigt mehr Platz in der Höhe und nutzt die in der Regel zur Verfügung stehende Breite von Desktop-Monitoren nicht aus.

Keine Variante: Labels in das Eingabefeld

Sie sollten auf keinen Fall die Beschriftungen **in** das Eingabefeld schreiben. Es existieren eine ganze Reihe Skripte, die Beschriftungen ausblenden und den Text der Beschriftungen als Platzhalter-Text in die `input`- oder `textarea`-Felder schreiben.

Was passiert wenn der Nutzer ein solches Feld ansteuert und fokussiert? Entweder verschwindet der Text und damit der einzige Hinweis, um was es in diesem Feld geht. Oder der Text bleibt stehen und der Nutzer muss ihn überschreiben. Dabei wird es passieren, dass viele Nutzer den Platzhaltertext abschicken, wie Untersuchungen zeigen. Damit dieser Platzhaltertext überhaupt von echten Eingaben unterscheidbar ist, muss der Designer ihn so hell anlegen, dass Sehbehinderte ihn nicht mehr erkennen können.

Vorhersagbare Platzierungen

Es ist eine Unsitte, Formularfelder aus reinen Platzgründen nebeneinander zu präsentieren, auch wenn diese in keinem inhaltlichen Zusammenhang zueinander stehen. Berechtigte Ausnahmen gibt es zwar bei natürlichen Kombinationen wie PLZ/Ort, Straße/Hausnummer oder Vorname/Nachname; allen anderen

Daten sollten Sie jedoch eine eigene Zeile zugestehen. Nutzer lernen beim ersten Betrachten eines Formulars dessen Aufbau und orientieren sich an diesem Gelernten. Wenn Sie dann mitten im Formular auf einmal von dieser Achse abweichen und rechts aussen zusätzliche Felder präsentieren, werden sie vom Nutzer leicht übersehen. Wenn es sich auch noch um Pflichtfelder handelt, führt dies auch zu einer Fehlermeldung und Frustration bei den Nutzern.

Gerade für Nutzer von Bildschirmen sind solche, je nach Vergrößerungsfaktor weit ausserhalb des sichtbaren Bereichs platzierte Formularfelder kaum zu finden, da niemand wohl nur »auf Verdacht« ganz nach rechts scrollen wird um nachzusehen, ob da vielleicht noch was ist.

Die horizontale Anordnung von Eingaben birgt auch weitere Gefahren in sich: durch die Abfolge von »Beschriftung/Eingabefeld, Beschriftung/Eingabefeld, Beschriftung/Eingabefeld ...« wird es praktisch unmöglich, Beschriftung und zugehöriges Eingabefeld korrekt zuzuordnen, wie im folgenden Screenshot deutlich wird.

Option 2 ☐ Option 3 ☐ Option 4 ☐ Option 4 ☐ Option 5 ☐ Option 6 ☐

Der vermeintliche Platzgewinn wird hier also durch ein Weniger an Eindeutigkeit erkaufte. Nutzer werden gezwungen, per »*Trial and Error*« herauszufinden, welche Option zu welchem Label-Text gehört. Die bessere Lösung wäre auch hier gewesen, die Optionen zeilenweise zu präsentieren.

Konventionen für Beschriftungstexte

Die meisten modernen Webbrowser bieten mittlerweile die Funktion, immer wiederkehrende Eingaben wie Vor- und Nachname oder Adresse zu speichern oder greifen auf das lokale Adressbuch des Nutzers zu. So kann der Nutzer dann per Knopfdruck entsprechende Felder automatisch ausfüllen lassen und erspart sich damit immer wiederkehrende Eingaben – für Nutzer mit motorischer Behinderung sicher eine große Hilfe.

Damit das funktioniert, müssen sich die Beschriftungen in den Formularen an bestimmte Konventionen halten. Daher sollten Sie die Beschriftungen so texten, wie die entsprechenden Einträge in Adressbüchern wie Outlook und ähnlichen lauten. Bewährt hat sich bei Adressen die Aufteilung in Vorname, Nachname, Firma, Straße, Hausnummer, Postleitzahl (bzw. PLZ), Ort (bzw. Stadt) und Land. Wenn Sie statt »E-Mail« »email« oder »Elektropost« als Beschriftungstext benutzen, dann mag das vielleicht witzig sein, aber kein Browser wird dieses Feld korrekt ausfüllen können.

Für die Schreibweisen und die Reihenfolge von Adressangaben gibt es sogar eine DIN-Norm (DIN 5008) – so kommt die Postleitzahl grundsätzlich vor dem Ort. Gerade eingedeutschte Anwendungen aus dem englischen oder amerikanischen Raum platzieren die Postleitzahl nach dem Ort und verstoßen damit gegen die Erwartung der Nutzer.

Ob Texte in Formularen und insbesondere in Labels fett gesetzt werden sollten, ist ebenfalls nicht eindeutig zu beantworten: Es gibt ernstzunehmende Hinweise, dass Nutzer fette Beschriftungen bevorzugen, weil sie einfacher zu scannen sind; allerdings gibt es auch Hinweise darauf, dass gefettete Texte schlechter lesbar sind:

»In their test, Penzo concluded that bold labels should be avoided if possible, as they are more difficult to read. However, our findings show a contradictory result. It conforms to Luke's recommendations where bold fonts can emphasis the labels from the foreground of the layout. Our participants found forms with bold labels easier to fill in.«

<zit>

Wie so oft hängt die Entscheidung hierüber vom konkreten Einzelfall ab, bei dem auch andere Aspekte wie die Auswahl des Zeichensatzes, das Farbschema oder die Abstände in Betracht gezogen werden müssen.

Konventionen für Pflichtfelder

Die Frage, ob optionale Felder oder Pflichtfelder gekennzeichnet werden sollten, lässt sich mit einer einfachen Regel beantworten: die Felder mit der geringeren Anzahl werden gekennzeichnet. Wenn Sie überwiegend Pflichtfelder haben, sollten Sie die wenigen optionalen Felder kennzeichnen. Umgekehrt gilt das gleiche: bei einer Vielzahl von Feldern oder bei komplexen Formularen, in denen aber nur wenige Pflichtfelder sind, ist eine Kennzeichnung der Pflichtfelder vorzuziehen. So verhindern Sie visuellen Lärm, überfrachten Ihr Formular nicht mit vielen einzelnen Informationen und die Inhalte werden dadurch für den Nutzer einfacher zu erfassen.

Keine der Varianten ist sinnvoll, wenn **alle** Felder Pflichtfelder sind. Diesen Umstand sollten Sie zu Beginn des Formulars erwähnen; auf keinen Fall erst am Ende des Formulars oder sogar nach dem »Absenden«-Button. Einen umgekehrten Fall gibt es hier nicht – zumindest ist uns bisher noch kein Formular untergekommen, in dem sämtliche Felder optional sind.

Für Anbieter ist es natürlich verführerisch, kurzerhand alle Felder zur Pflicht zu erklären, um an möglichst viele Daten zu gelangen. Dem gegenüber stehen die Ansprüche der Nutzer und des Datenschutzes, möglichst wenige Daten preiszugeben, um ein Ziel zu erreichen, sei es eine Registrierung, ein Einkauf oder eine Meinungsäußerung per Kommentar. Daher könnte man fordern, sämtliche optionalen Felder gleich ganz wegzulassen.

Die Realität liegt aber, wie so oft, zwischen diesen beiden Extremen: aus Sicht eines Anbieters kann es durchaus sinnvoll sein, bestimmte Daten des Nutzers zu erfahren (z.B. wie er zur Website gekommen ist), aber sicher nicht alle Nutzer sind bereit, solche Angaben zu machen. Da würde die Festlegung als Pflichtfeld sicher unnötige Abbrüche provozieren. Um hier nicht unnötig Kunden zu verlieren, sollten solche nicht-essentiellen Felder also optional sein.

Zur Erklärung, dass Pflichtfelder enthalten sind, die zwingend ausgefüllt werden müssen, gehört die Erklärung, wie diese gekennzeichnet sind. Die hierfür häufig verwendeten Asterisken (*) haben das Problem, dass diese von Screenreadern je nach Einstellungen des Nutzers als Interpunktionszeichen behandelt und nicht vorgelesen werden. Um auf der sicheren Seite zu sein, benutzen wir hier bei »Einfach für Alle« Grafiken mit einem Sternchen, in denen der Alternativ-Text "Pflichtfeld" hinterlegt ist. Zusätzlich können Sie das Sternchen auch mit einem title-Attribut ausstatten – wegen eines unnötigen Dopplers brauchen Sie sich hier keine Sorgen zu machen, Screenreader lesen immer nur *entweder alt oder title* vor (Testdatei):

Ein sehr spezifisches Problem betrifft gehörlose Nutzer: Sie können in der Regel mit einem Telefonanruf nicht viel anfangen, sondern bevorzugen die schriftliche Kommunikation via E-Mail oder SMS. Daher sollten Telefonnummern nicht als Pflichtfelder hinterlegt werden, oder zumindest mit einer zusätzlichen Checkbox versehen sein, wo der Nutzer angeben kann dass er die Kontaktaufnahme via SMS bevorzugt.

Was geht gar nicht?

Ein immer wieder gern gemachter Fehler ist, Pflichtfelder ausschließlich optisch zu kennzeichnen, z.B. durch farbige Markierungen per CSS (Hintergrundfarben, Ränder etc.). In HTML-Code wie dem Folgendem steht dann für alternative Ausgabeformen nichts, aber auch wirklich gar nichts an verwertbarer Information:

```
<div class="pflichtfeld"><input ...></div>
```

Hier bleibt dem Nutzer einer Sprachausgabe nichts anderes übrig als es auf gut Glück zu versuchen.

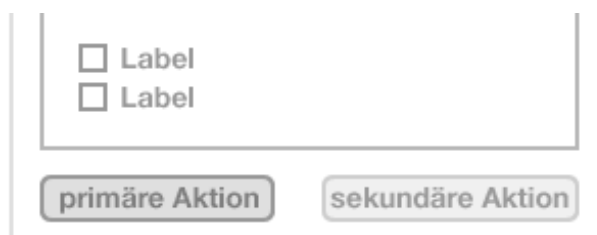
Wohin mit der Kennzeichnung?

Die Kennzeichnung der Pflichtfelder sollte auf jeden Fall mit den Beschriftungen verknüpft sein bzw. innerhalb des Label-Elements stehen. Ob Sie die Kennzeichnung optisch links oder rechts der Kontroll- und Eingabeelemente platzieren, spielt in Zeiten von CSS keine Rolle mehr: mit dem nötigen Geschick können Sie die Kennzeichnung auch aus einem links stehenden Label herausziehen und rechts neben den Feldern platzieren. Wichtig ist nur, dass die Kennzeichnungen konsistent immer auf einer gemeinsamen erkennbaren Achse und an einer vorhersagbaren Position stehen und somit schneller zu erfassen sind.



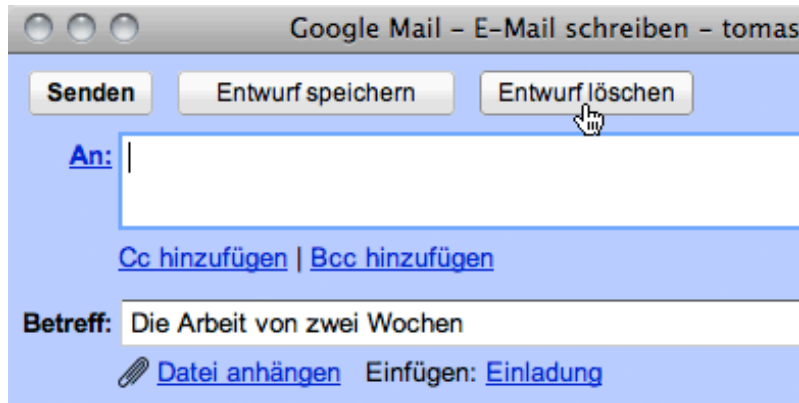
Konventionen für Buttons

Auch für Buttons gibt es Konventionen, an denen sich Ihr Formulardesign orientieren sollte. Die wichtigste: die primäre Aktion, also in den meisten Fällen wohl »Speichern« oder »Abschicken«, steht immer an der prominentesten Stelle – sie ist direkt verantwortlich für die Fertigstellung des Vorgangs und die Präsentation sollte ihrer Wichtigkeit entsprechen. Sekundäre Aktionen wie »Abbrechen«, »Löschen« und »Zurück« sind nur selten sinnvoll, weil sie in der Regel den Zielen des Formulars zuwider laufen. Wenn Sie diese trotzdem einsetzen, dann sollte die optische Gewichtung ihrer (potenziell destruktiven) Funktion entsprechen, wie dies generell in Anwendungen jenseits des Browsers der Fall ist:



Auch wenn das Ziel eines guten Formulardesigns sein sollte, den Nutzer möglichst schnell durch den Prozess zu geleiten – hier sollte der Nutzer gebremst werden. Er soll sich mit der Beschriftung der Buttons befassen, bevor er wahllos etwas drückt, was annähernd so aussieht wie ein Button. Wie Luke Wroblewski

im Artikel »Primary & Secondary Actions in Web Forms« anhand von Nutzertests nachweist, führt eine gleichartige Präsentationsform von »Speichern« (`<input type="submit">`) und »Löschen« (`<input type="reset">`) zu unverträglich hohen Fehlerquoten und in Folge zu Frustration bei den Nutzern, die durch diese Verwechslung sämtliche Eingaben verlieren. Eine Möglichkeit, dies zu verhindern ist, die potenziell zerstörerischen Aktionen abweichend zu gestalten und z.B. in der Optik von Links statt von Buttons zu präsentieren und diese nicht, wie im folgenden Screenshot einer Webmail-Anwendung, gleichrangig nebeneinander zu stellen:



Eine gute Zusammenfassung zur Diskussion um die richtige Platzierung und das Aussehen von primären und sekundären Aktionen finden Sie in den Artikeln »The use of buttons in web forms« von Gabriel Svennerberg und »Buttons on Forms – where to put them, and what to call them« von Caroline Jarrett. Wie bei der überwiegenden Anzahl der Experten wird auch hier die Platzierung der primären Aktion auf der linken Seite empfohlen, die sekundären Aktionen sollten rechtsbündig ausgerichtet werden.

Fangen spielen?

Ebenso wichtig wie die Benennung und Platzierung ist, gerade aus Sicht der Barrierefreiheit, die Größe von Interface-Elementen wie Buttons, aber auch Icons und damit der Klickfläche, die ein Nutzer treffen kann. Eines der fundamentalen Gesetze des Interaktionsdesigns ist das nach seinem Entdecker benannte Fitts's Law:

$$t = a + b \log_2 (D / B + 1)$$

- wobei t die Zeit ist, die benötigt wird, um das Ziel zu treffen, D ist die Distanz vom Ausgangspunkt des Cursors und B ist die Breite des Ziels. Oder etwas salopper formuliert: je größer ein Scheunentor ist und je näher man davor steht, desto schneller trifft man das Scheunentor mit einer Schrotflinte. Eine hervorragende Einführung in diese Gesetzmäßigkeiten finden Sie im Artikel »Visualizing Fitts's Law« von Kevin Hale.

Gerade bei der aktuell stark zunehmenden mobilen Nutzung mit Touchscreens gewinnt dieses Gesetz an Relevanz: ein Interface-Element, das nicht mindestens 40-50px groß ist, kann vom menschlichen Zeigefinger nicht verlässlich getroffen werden – besonders wenn in unmittelbarer Nähe weitere gleich kleine Elemente stehen.

Ergebnisse zählen

In der Konzeption Ihrer Formulare sollten Sie auch den abschließenden Bestätigungs- oder Ergebnisseiten die angemessene Aufmerksamkeit widmen. Gerade bei vertriebsorientierten Seiten bietet sich hier ein Potenzial, den Prozess für den Kunden mit einem angenehmen Erlebnis abzuschließen. Dazu gehört auch die Erstellung einer für den Ausdruck optimierten Fassung der Seite mit einer Zusammenfassung der übermittelten Daten.

3. Technik: HTML & CSS für Formulare

Semantik in Formularen – welche Elemente wofür in HTML4 /

XHTML1?

Die Menge und Möglichkeiten der in HTML4 und XHTML1 vorhandenen Kontrollelemente in Formularen sind begrenzt, vor allem im Vergleich mit den Möglichkeiten graphischer Betriebssysteme. Sie haben allerdings auch Vorteile:

- Ihr Zweck und ihre Funktionen sind eindeutig definiert.
- Sie werden in der Regel auch von älteren Browsern zuverlässig unterstützt.
- Sie sind mit den üblichen Hilfsmitteln behinderter Nutzer gut zu bedienen.

Welche Formularelemente gibt es?

Die folgenden Formularelemente gibt es in HTML

- einzeilige und mehrzeilige Textfelder zur Eingabe (INPUT mit dem Attribut `type="text"` sowie mit dem Attribut `type="password"` für Passwörter und TEXTAREA)
- Elemente für Datei-Uploads (INPUT mit dem Attribut `type="file"`)
- Auswahllisten (SELECT und OPTION),
- der Sonderfall Auswahllisten mit Mehrfachauswahl (SELECT mit dem `multiple`-Attribut und OPTION)
- Radiobuttons zur Entweder/Oder-Auswahl (`<input type="radio">`)
- Checkboxes zur Sowohl/Als auch-Auswahl (`<input type="checkbox">`)
- sowie die Schaltflächen-Elemente (Buttons) zum Übergeben von Daten an den Server und zum Löschen von Formularen:
 - `input type="submit"`
 - `input type="reset"`
 - `input type="image"`
 - `input type="button"` sowie:
 - `button`

Im Sinne der Barrierefreiheit sind alle diese Elemente neutral. Ihr Einsatz stellt keine besonderen Hürden im Umgang mit Web-Formularen dar. Allerdings gibt es zwei Ausnahmen:

- Der Button zum Zurücksetzen eines Formulars auf den ursprünglichen, meist leeren Zustand (`<input type="reset">`) ist selten sinnvoll. Der Schaden, den ein solcher Button anrichten kann, überwiegt bei weitem dessen Nutzen. Seine ganze fatale Wirkung entfaltet dieser Button, wenn »Abbrechen« vor der primären Aktion steht (also üblicherweise »Absenden« oder »Speichern«): Ein unachtsamer Druck auf Enter leert unter Umständen das Formular und befördert alle Eingaben des Nutzers zuverlässig in den Datenhimmel. Erschwerend kommt hinzu, dass je nach verwendetem Betriebssystem des Nutzers unterschiedliche Konventionen zur Platzierung des »Abbrechen«-Buttons herrschen. Man kann also nie vorhersagen, ob Nutzer diesen Button links oder rechts erwarten. Die Wahrscheinlichkeit, dass Nutzer diesen Button irrtümlich drücken ist also recht hoch – zu hoch um den Einsatz zu rechtfertigen.
- Die eingangs der Auflistung erwähnten Elemente zur Mehrfachauswahl (`<select size="10" multiple="multiple">`): Bei diesen Feldern ist für den Benutzer nicht ohne weiteres erkennbar, dass hier tatsächlich mehrere Optionen ausgewählt werden können – damit hätten wir die erste kognitive Barriere.

Fleißige Helferlein: Formular-Frameworks & -Generatoren

Große Frameworks wie YUI Grids oder YAML mit dem YAML Builder stellen oft eigene Anwendungen zur Erstellung von HTML & CSS-Gerüsten für komplette Websites zur Verfügung. Es gibt aber auch eine ganze Reihe kleinerer Anwendungen, die auf die Erstellung von Formularen spezialisiert sind. Dazu gehören zum Beispiel:

- Formy
- Formee
- Uni-Form
- pForm
- Wufoo
- FormAssembly
- Formalize CSS
- Uniform

Zudem ist die Bedienung (Auswahl per Maus mit gedrückter STRG- bzw. Apfel-Taste) nicht intuitiv und kann nicht als gelernt vorausgesetzt werden – eine weitere Barriere. Zudem lässt sich dieses Element nur umständlich oder im Ernstfall gar nicht per Tastatur bedienen. Bevor Sie sich hier in langatmigen Hilfetexten verzetteln, sollten Sie gleichwertige Checkboxes für Mehrfachauswahlen einsetzen, wenn deren Anzahl ein vernünftiges Maß nicht übersteigt.

Welche Formularelemente gibt es (noch) nicht?

Für die meisten klassischen Formulare wird der begrenzte Satz von Formularelementen von HTML4 und XHTML1 ausreichen. Bei web-basierten Anwendungen stößt man jedoch sehr schnell an die Grenzen der Markup-Sprache, die eigentlich nie für solche Anwendungen gedacht war. Was in den fertig verabschiedeten HTML-Standards bis heute fehlt sind Elemente, die jenseits des Web-Browsers gang und gäbe sind, wie zum Beispiel:

- Schieberegler (Slider),
- Numeric Stepper (Textfelder, in denen mit Pfeiltasten die Werte herauf- oder herabgesetzt werden können),
- Comboboxen mit Mehrfachauswahl und eigenen Eingaben
- Fortschrittsbalken
- Farbauswahl
- Datumsauswahl
- Checkboxes mit drei Zuständen (Tri-State: Ja, Nein und Teilweise)
- u.v.m. ...

Hier soll in Zukunft HTML5 Abhilfe schaffen, das einen erweiterten Satz an Eingabe- und Kontrollelementen zur Verfügung stellen wird. Zum Zeitpunkt der Veröffentlichung dieser Artikelserie befand sich HTML5 aber immer noch im Zustand eines Entwurfes, der in aktuellen Browsern teilweise schon implementiert ist. Da es sich aber um einen Working Draft handelt, kann sich immer noch vieles ändern. Ein Einsatz der neuen Elemente sollte also erst nach genauer Abwägung der Vor- und Nachteile und nach eingehenden Tests geschehen.

Falls Ihre Anwendung diese Elemente bzw. deren Funktionalität benötigt, so können Sie für moderne Browser die neuen Elemente aus HTML5 auch heute schon verwenden. Allerdings benötigen nicht ganz so moderne Browser und Hilfsmittel Fallback-Lösungen und/oder JavaScript, um die gewünschten Funktionen zur Verfügung zu stellen.

Teilweise wird dieser Spagat durch den Einsatz von WAI-ARIA erleichtert, das als eine Art Spachtelmasse die Lücke zwischen der begrenzten Welt von HTML4 und den erweiterten Möglichkeiten von HTML5 zu überbrücken sucht. Hierzu mehr in einem späteren Kapitel der Serie.

Welches Element ist das Richtige?

Bei der Festlegung der Kontrollelemente sollten Sie sich folgende Fragen stellen:

- Was ist für den Nutzer nahe liegender – das Eintippen eines freien Textes oder die Auswahl aus einer Liste von Optionen? Was fängt die möglichen Eingaben des Nutzers am besten ab? So kann man in einem Bestellvorgang die Anzahl der zu bestellenden Produkte als Textfeld oder als Auswahllisten (z.B. 1-10) hinterlegen. Im Zweifelsfall sollten Sie sich hier für die größtmögliche Freiheit für den Nutzer entscheiden. Ein Beispiel: Bei »Einfach für Alle« gab es früher eine Broschüre, die über ein Bestellformular angefordert werden konnte. In diesem Bestellformular war die Anzahl der Broschüren über ein Textfeld frei einzugeben; zur Verhinderung von Scherzbestellungen war dieses Textfeld per `maxLength`-Attribut auf zwei Ziffern begrenzt. Nun gab es aber tatsächlich Fälle, in denen jemand gleich einen ganzen Karton mit 100 Broschüren für eine Bildungseinrichtung bestellen wollte, was mit diesem Formular unmöglich war.
- Mit welchem Element kann der Nutzer die wenigsten Fehler machen? Kann man sich einfach vertippen? Dann sollten Sie bei einer überschaubaren Anzahl von Optionen dem Nutzer die Arbeit abnehmen und diese als Auswahl präsentieren. Bei der Eingabe von freien Texten können Sie den Nutzer unterstützen,

indem die Eingaben unmittelbar clientseitig validiert bzw. im Hintergrund an den Server geschickt werden, um die Plausibilität der Eingaben zu überprüfen.

- Muss der Nutzer sämtliche Optionen lesen, um die Abfrage zu verstehen? Dann sollten Sie die beschreibenden Texte und Labels möglichst kurz und prägnant halten. Die Erfahrung hat gezeigt, dass Nutzer umso weniger Text lesen, je mehr in einem Formular steht. In Tests ist sogar oft zu beobachten, dass Nutzer noch während des Ladens der Seite anfangen, mit dem Formular zu interagieren, ohne sich vorher in Ruhe zu orientieren und eventuell bereitgestellte Hilfen zur Kenntnis zu nehmen.
- Wie viele Optionen gibt es? Kann der Nutzer mehr als eine Option auswählen? Bei begrenzten Auswahlmöglichkeiten oder wenn immer nur eine Option zulässig ist (wie die Angabe der Kreditkarte in einem Bezahlvorgang) fällt schon fast zwangsläufig die Wahl auf Radiobuttons. Bei Mehrfachauswahlen wiederum bieten sich Checkboxes an.
- Kann man die zur Wahl stehenden Optionen als bekannt voraussetzen oder geht es um die Auswahl aus einer Liste, die dem Nutzer unbekannt sein dürfte? Bei einfachen Fragen, die mit Ja oder Nein zu beantworten sind, fällt die Wahl natürlich auf Radiobuttons – hier hat der Nutzer alle Optionen im direkten Zugriff. Bei längeren Listen (z.B. eine Auswahl aus den 16 Bundesländern) macht es keinen Sinn, diese allesamt als eine Liste von Radiobuttons zu zeigen – diese Auswahl sollte als SELECT gezeigt werden. Weitere Gedanken zu diesem Thema finden Sie im Artikel »It's not about size, it's about context – radio buttons or drop-downs« von Donna Spencer.
- Wollen Sie Ihren Benutzern wirklich die Navigation in einer Auswahlliste mit mehreren hundert Einträgen zumuten? Schlechte Beispiele hierfür finden sich im Netz zuhauf bei den Länderauswahlen: viele Anbieter hinterlegen hier sämtliche Staaten der Erde, auch wenn sie ihre Waren nur in Deutschland und dem benachbarten Ausland ausliefern. Auch die Strukturierung nach Kontinenten per OPTGROUP ist nur dann wirklich sinnvoll, wenn Sie tatsächlich Kunden auf allen Kontinenten bedienen wollen. Die Beschränkung auf eine Anzahl realistischer Optionen ist hier oftmals sinnvoller.
- Was entspricht den (Seh-) Gewohnheiten der Nutzer? Bei der Auswahl eines Datums (nach der Art TT MM JJJJ) werden immer wieder unbeabsichtigte Barrieren aufgebaut. Bei der freien Eingabe in ein Textfeld kann es zu Problemen mit der Internationalisierbarkeit (I18N) kommen, wenn Nutzer aus unterschiedlichen Ländern unterschiedlichen Konventionen in der Sortierung folgen. Für einen Europäer ist 01/04/07 der 1. April 2007, für Amerikaner wäre dies jedoch der 4. Januar 2007; in Ländern mit umgedrehter Leserichtung (von rechts nach links) könnte dies auch als der 7. April 2001 gelesen werden. Auch die Hinterlegung als getrennte Auswahl in Form von getrennten Auswahllisten für Tag, Monat und Jahr ist suboptimal, da nur umständlich zu bedienen. Das natürlichste Element mit dem geringsten Verwechslungspotenzial wäre hier ein Kalender, den man mit ein wenig JavaScript und einer Datentabelle ins Formular integrieren kann. Wie dies barrierefrei zu realisieren ist zeigt das folgende Beispiel: »Unobtrusive JavaScript date-picker widget«

Logische Blöcke: Fieldset & Legend

Wenn Sie die Komplexität auf das Notwendige reduziert haben, geht es nun an die Aufteilung der Oberfläche. Optische Gruppierungen sollten die strukturellen Gruppierungen unterstützen; mit Überschriften bzw. Legenden, Farben und Formen können Sie Bereiche des Formulars ordnen, die dann vom Besucher nacheinander abgearbeitet werden. So können Sie unerfahrenen Benutzern die einzelnen Schritte schrittweise oder seitenweise präsentieren und diese mit erklärenden Texten unterbrechen.

Wenn Sie ein komplexes Formular anbieten, das aus mehreren logischen Einheiten besteht, sollten Sie über den Einsatz des FIELDSET-Elementes nachdenken. Mit einem Fieldset lassen sich Formulare in leichter überschaubare Blöcke unterteilen. Es werden genau die Bestandteile zusammengefasst, die eine logische Einheit ergeben. Diese können dann übersichtlicher gestaltet und vom Besucher nacheinander abgearbeitet werden. So können Sie unerfahrenen Benutzern die einzelnen Schritte häppchenweise präsentieren und diese mit erklärenden Texten (»Geben Sie bitte Ihre persönlichen Daten ein...«) unterbrechen.

XHTML1 ist nicht nur HTML 4 mit ein paar zusätzlichen Schrägstrichen, es gibt auch tiefergehende Unterschiede. LEGEND ist ein Beispiel hierfür: In HTML ist das Element als direktes Kind-Element von FIELDSET zwingend

Gerade bei Formularen, in denen die Eingabe einer Vielzahl unterschiedlicher Daten verlangt wird, trägt diese Technik zur Übersichtlichkeit für den Anwender bei. Für Nutzer, bei denen die Hürden weitestgehend im kognitiven Bereich bestehen, können Sie so die Zahl der Abbrecher reduzieren.

Durch den Einsatz von Legenden für die einzelnen Fieldsets können Sie Ihren Besuchern eine weitere Orientierungshilfe geben. Das Element `LEGEND` ist ein Kind-Element von `FIELDSET`, das von grafischen Browsern am Beginn des abgegrenzten Bereichs dargestellt wird. Per CSS und JavaScript lassen sich hier Strukturen schaffen, die zum Beispiel eine Präsentation in Form von Karteireitern und -karten ermöglichen:

The image shows a web form with three steps: '1. Schritt', '2. Schritt', and '3. Schritt'. The first step is active and contains the following elements:

- A text input field with the label 'Label'.
- A dropdown menu with the label 'Auswahl' and a downward arrow.
- Two checkboxes, each with the label 'Label'.

Fieldsets können übrigens auch verschachtelt werden, d.h. innerhalb eines Fieldsets können weitere Fieldsets enthalten sein (Beispieldatei: `fieldsets.html`), um zum Beispiel zusammengehörige Gruppen von Kontrollelementen darzustellen:

vorgeschrieben, in XHTML1 ist `LEGEND` hingegen optional.

Aus der HTML 4.01 Strict-DTD:

```
<!ELEMENT FIELDSET - -
(#PCDATA,LEGEND,(%flow;))* -
- formcontrol group -->
```

und der korrespondierende Eintrag aus der XHTML-DTD:

```
<!ELEMENT fieldset (#PCDATA |
legend | %block; | form |
%inline; | %misc;)*>
```

Leider sind eine Reihe dieser Unterschiede in der offiziellen Dokumentation des W3C nicht beschrieben, was sicher auch einer der Gründe für die sinnfreie Fehlermeldung des W3C-Validators in diesem Punkt ist. Zum Vergleich: `fieldset-ohne-legend.xhtml` vs. `fieldset-ohne-legend.html`

```

<fieldset>
  <legend>Äussere Legende</legend>
  <label for="textinput1">Explizites Label:</label><br>
  <input type="text" name="textinput1" id="textinput1"><br>
  <label>Implizites Label:</label><br>
  <input type="text" name="textinput2" id="textinput2"></label><br>

  <fieldset>
    <legend>Innere Legende Eins</legend>
    <input type="radio" name="flavor1" id="toffee" value="toffee">
    <label for="toffee">Vanilla Toffee Crunch</label><br>
    <input type="radio" name="flavor1" id="chunky" value="chunky">
    <label for="chunky">Chunky Monkey</label><br>
    <input type="radio" name="flavor1" id="oatmeal" value="oatmeal">
    <label for="oatmeal">Oatmeal Cookie Chunk</label>
  </fieldset>

  <fieldset>
    <legend>Innere Legende Zwo</legend>
    <input type="checkbox" name="flavor2" id="brownie" value="chocolate">
    <label for="brownie">Chocolate Fudge Brownie</label><br>
    <input type="checkbox" name="flavor2" id="cookie" value="cookie">
    <label for="cookie">Chocolate Chip Cookie Dough</label><br>
    <input type="checkbox" name="flavor2" id="fudge" value="fudge">
    <label for="fudge">New York Super Fudge Chunk</label>
  </fieldset>

  <button type="submit">Haben Wollen!</button>
</fieldset>

```

In der grafischen Ausgabe sind diese Fieldsets deutlich voneinander getrennt; per CSS können Sie nun den inneren Fieldsets ein anderes Aussehen verpassen als dem Äusseren:

Diese Verschachtelungen können Sie sich beispielsweise zunutze machen und optionale Formularbereiche gruppieren, die vom Nutzer nicht zwingend ausgefüllt werden müssen. Der Nutzer kann diese so einfacher überspringen; per JavaScript können Sie den Nutzer diese optionalen Felder auch einfach ausblenden lassen. Mit FIELDSET haben Sie hier das semantisch naheliegendste Element gewählt statt eines bedeutungslosen generischen DIV.

Zu viel des Guten: Überlange oder unnötige Legenden

Verbreitete Screenreader lesen im Formularmodus den Inhalt einer Legende nicht zu Beginn des Fieldsets vor. Stattdessen wird der Text bei jedem einzelem Kontrollelement bzw. dessen Label vorgelesen, das sich innerhalb des damit ausgezeichneten Fieldsets befindet. Dies kann, je nach Legendentext, sehr schnell dazu führen, dass Ihre Formulare im Screenreader kaum noch zu benutzen sind. Beim Nutzer werden ganz sicher Ermüdungserscheinungen hervorgerufen, wenn er gezwungen wird, sich den immer gleichen Zusatztext bei jeder einzelnen Checkbox und bei jedem einzelem Radiobutton anzuhören.

Das Verhalten der Screenreader ist an sich nicht schlecht: so wird dem Nutzer jederzeit die Orientierung geboten, in welchem Bereich eines Formulars er sich befindet – was dem sehenden Nutzer durch optische

Zusammenhänge klar wird, in der Sprachausgabe jedoch naturgemäß fehlt. Problematisch wird dieses Verhalten nur, wenn der Legendentext eine bestimmte Länge überschreitet oder im gesprochenen Zusammenhang mit dem Labeltext keinen Sinn mehr ergibt.

Wann Sie diese Grenze überschritten haben, ab der es zu viel wird, können Sie mit einem einfachen Test auch ganz ohne Screenreader feststellen: Lesen Sie den Textinhalt des Formulars laut vor und setzen die Legende erneut vor jedes Label. Wenn sich das dann so anhört:

»Ihre persönlichen Daten eingeben Ihr Vorname Ihre persönlichen Daten eingeben Ihr Nachname Ihre persönlichen Daten eingeben Ihre Adresse Ihre persönlichen Daten eingeben Ihre Postleitzahl Ihre persönlichen Daten eingeben Ihr Ort Ihre persönlichen Daten eingeben Abschicken«

<zitat>

dann haben Sie klaren Bedarf zur Nachbesserung. Unter Umständen kann es angebracht sein, auf LEGEND ganz zu verzichten, besonders wenn die Kontrollelemente durch die verwendeten Labels ausreichend beschrieben sind. Leider geben einige Accessibility-Checker einen Fehler aus, wenn kein LEGEND im Code vorgefunden wird. Unsere kritische Haltung gegenüber diesen Prüfprogrammen ist hinreichend bekannt – im Ernstfall sollte man sich hier für den Nutzer entscheiden.

Ein weiteres Problem entsteht, wenn verbreitete Screenreader wie JAWS die Zuordnung der LEGEND zu den Formularelementen kurzerhand selber übernehmen und anfangen zu raten. Im nächsten Codelisting sehen Sie eine durchaus zulässige Art der Verschachtelung, die aber bei Screenreader-Nutzern für einige herz hafte Lacher sorgen dürfte:

```
<fieldset>
  <legend>Persönliche Informationen</legend>
  <label for="vorname">Vorname</label>
  <input type="text" id="vorname" name="vorname">
  <label for="nachname">Nachname</label>
  <input type="text" id="nachname" name="nachname">

  <fieldset>
    <legend>Geschlecht</legend>
    <input type="radio" id="mann" name="geschlecht" value="mann">
    <label for="mann">Mann</label>
    <input type="radio" id="frau" name="geschlecht" value="frau">
    <label for="frau">Frau</label>
  </fieldset>

  <label for="telefon">Telefon</label>
  <input type="text" id="telefon" name="telefon">
</fieldset>
```

Hier wird JAWS die Legende »Persönliche Informationen« für die beiden Textfelder »Vorname« und »Nachname« übernehmen und als nächstes dann »Geschlecht Mann Geschlecht Frau Geschlecht Telefon« vorlesen, weil »Geschlecht« die von »Telefon« aus gesehen nächste Legende ist. Weitere interessante Hintergrundinfos über das Verhalten verbreiteter Screenreader finden sie in dem Artikel »Fieldsets, Legends and Screen Readers« von Steve Faulkner

Sind Überschriften nicht manchmal besser?

Wenn Ihr Formular in einer ansonsten mit Überschriften strukturierten (Text-) Seite steht, kann es durchaus sinnvoll sein, auf den Einsatz von Fieldset und Legend ganz zu verzichten und dem Formular stattdessen eine eigene Überschrift (H1-6) zu geben. Auch wenn Ihr Formular aus einem einzigen Kontrollelement besteht, ist das volle Programm aus FIELDSET, LEGEND usw. oftmals ein Code-Overkill, mit dem bestenfalls redundante Informationen mehrfach transportiert werden.

Strukturen im Kleinen: Optgroup

Ein Element fristet leider ein Schattendasein und wird im Code von Formularen viel zu wenig eingesetzt: OPTGROUP. So wie Sie mit Fieldsets größere logische Einheiten von Formularelementen gruppieren können, so geht dies per <optgroup>-Tags auch innerhalb von Auswahlfeldern vom Typ SELECT. Innerhalb einer Optgroup können zum Beispiel Länder der jeweiligen Kontinente zusammengefasst werden oder Geschmacksrichtungen bei der Auswahl von Eissorten, wie das folgende Markup-Beispiel zeigt:

```
<select name="Eissorte">

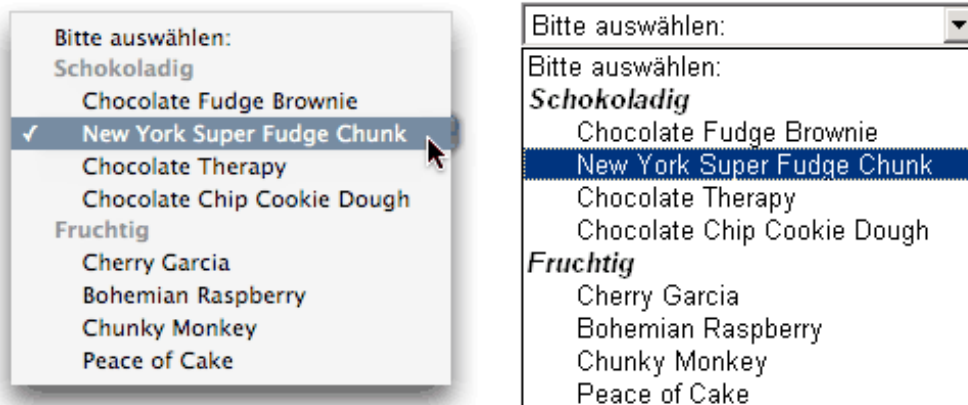
  <option disabled selected>Bitte auswählen:</option>

  <optgroup label="Schokoladig">
    <option>Chocolate Fudge Brownie</option>
    <option>New York Super Fudge Chunk</option>
    <option>Chocolate Therapy</option>
    <option>Chocolate Chip Cookie Dough</option>
  </optgroup>

  <optgroup label="Fruchtig">
    <option>Cherry Garcia</option>
    <option>Bohemian Raspberry</option>
    <option>Chunky Monkey</option>
    <option>Peace of Cake</option>
  </optgroup>

</select>
```

Durch das label-Attribut von OPTGROUP (nicht zu verwechseln mit dem LABEL-Element!) können nun den einzelnen Gruppen noch Kennzeichnungen vorangestellt werden, die selbst nicht auswählbar sind, sondern nur der Strukturierung dienen. Das Ergebnis sieht im Browser wie folgt aus:



Etikettierung mit Label

Neben der Navigation sind Formulare die Elemente einer Webseite, mit denen die Nutzer am häufigsten interagieren. Daher ist gerade bei Formularen nicht nur die Unabhängigkeit von den Ausgabemedien, sondern auch von den Eingabemedien wichtig. Eine wirkliche Barrierefreiheit erreichen Sie erst, wenn die bereitgestellten Funktionen mit einer Vielzahl von Eingabe- und Zeigegeräten bedient werden können. Zum Glück bietet HTML hierfür Elemente, die eine Bedienung unabhängig von der Umgebung ermöglichen.

Sie kennen diese Techniken aus Ihrem Betriebssystem und anderen Anwendungen: in Dialogen mit Checkboxes und Radiobuttons lassen sich nicht nur die Kontrollelemente selbst betätigen; es reicht auch, wenn man den daneben stehenden Text anklickt und schon ist das Häkchen gesetzt. Der Hintergrund dafür ist eine einfache Regel: je größer das Ziel, desto höher die Trefferwahrscheinlichkeit.

Mit dem dafür nötigen kleinen Kunstgriff kommen nicht nur erfahrene Nutzer schneller zum Ziel, die diese

größere Klickfläche aus Anwendungen gewohnt sind. Gerade Menschen mit motorischen Behinderungen, die Probleme mit der exakten Positionierung der Maus haben oder alternative Zeigegeräte benutzen, können nun auf den dazugehörigen Text des Formularfeldes, das sogenannte Label, klicken. Blinde oder stark sehbehinderte Nutzer von Screenreadern sind ebenfalls auf die Bedienung per Tastatur angewiesen. Erst durch die Verknüpfung des Kontrollelementes mit seiner Beschriftung wird in der Sprachausgabe Sinn und Zweck des jeweiligen Kontrollelementes klar.

Label in HTML

HTML sieht hierfür seit der Version 4 das Element LABEL vor. Über dieses Label können Sie die Feldbeschriftung mit dem dazu gehörigen Kontrollelement verknüpfen, indem Sie letzterem eine eindeutige ID (id="textfeld1") zuweisen. Diese ID wird dann ebenso im for-Attribut des Start-Tags von LABEL benutzt und schon ist die Beschriftung mit dem jeweiligen Formularelement verknüpft und anklickbar:

```
<label for="textfeld1" class="left">Text:</label>
<input id="textfeld1" type="text">
[...]
```

```
<input id="checkbox1" type="checkbox" value="ja">
<label for="checkbox1">Ja</label>
```

Diese Methode funktioniert bei allen Kontrollelementen, denen ein Label zugewiesen werden kann. Beachten Sie, dass einem Kontrollelement gemäß dem HTML-Standard zwar mehrere Label zugeordnet werden können, dies aber zu Problemen führt (Beispieldatei: multiple-label.html).

Folgenden Elementen können und sollten Label zugewiesen werden:

- input type="text"
- input type="checkbox"
- input type="radio"
- input type="file"
- input type="password"
- textarea
- select

Label können nicht für folgende Elemente verwendet werden, da hier die Informationen über das value-Attribut (z.B. für submit-Buttons), das alt-Attribut (für grafische Buttons vom Typ input type="image") oder den Inhalt des Elementes selbst transportiert werden:

- Submit und Reset-Buttons (input type="submit" bzw. input type="reset")
- Grafische Buttons (input type="image")
- Versteckte Eingabefelder (input type="hidden")
- Script-Buttons (BUTTON-Element bzw. input type="button")

Warnhinweise: Probleme mit Screenreadern

Implizite Label

Nach den Spezifikationen ist es zulässig, das Label um das Kontrollelement herum zu legen, statt es ihm vor- oder nachzustellen. Hiermit könnte man sich als Webentwickler die Zuweisung über id="" und for="" sparen, da das Label nun durch diese Verschachtelung implizit zugeordnet ist:

```
<label>Text:<input type="text"></label>
```

Soweit die Theorie. Die Unterstützung für diese Variante ist jedoch selbst in aktuellen assistierenden Programmen eher lückenhaft und daher nicht zu empfehlen. Selbst aktuelle Versionen der verbreiteten Screenreader JAWS und Window Eyes haben bis dato Probleme mit impliziten Labels, obwohl diese bereits seit 1997 Bestandteil der HTML 4-Empfehlung sind. So werden implizite Labels für Checkboxes und Radiobuttons sogar im Formularmodus nicht vorgelesen, bei Textfeldern ist dies nach Aussage von Nutzern

»Glückssache««. Um maximale Kompatibilität zu garantieren, sollten Sie auf die zuerst beschriebene Form der expliziten Verknüpfung zurückgreifen.

Falls Sie aus irgendwelchen Gründen gezwungen sein sollten, implizite statt explizite Labels zu benutzen, so können Sie den Label-Text in einem `title`-Attribut des jeweiligen Kontrollelementes wiederholen. In diesem Fall lesen JAWS, Window Eyes & Co. den Text des `title`-Attributs, sobald das Kontrollelement den Fokus erhält.

Mehrere Label zuweisen

Oft besteht die Notwendigkeit, einem Kontroll- oder Eingabeelement mehrere Label zuzuweisen. Dies ist zum Beispiel dann der Fall, wenn ein Fehler aufgetreten ist und man neben dem ursprünglichen Label auch noch die Fehlermeldung mit dem Ort des Fehlers verknüpfen will. Was liegt da näher als diese Fehlermeldung gleich als Label zu deklarieren – wie sich beim Blick in die HTML4-Spezifikation herausstellt eine durchaus zulässige Methode:

»Mehr als ein LABEL kann mit dem gleichen Steuerelement verknüpft werden, wenn mehrere Verweise über das `for`-Attribut erzeugt werden.«

<zitat>

In grafischen Browsern klappt das auch hervorragend: beide Labels lassen sich anklicken und die verknüpften Formularelemente werden fokussiert. Nur leider tanzen hier mal wieder einige Screenreader aus der Reihe:

- VoiceOver von Apple erkennt nicht mehr als ein Label und liest immer das Label vor, das im Quelltext zuerst steht.
- NVDA liest in der Kombination mit Firefox ebenfalls nur das erste Label vor.
- JAWS und Window Eyes mit Internet Explorer machen genau das Gegenteil und lesen nur das letzte Label vor.

Mehrere Labels sollten also nur nach intensiver Abwägung und Tests eingesetzt werden, gegebenenfalls müssen auch die Labeltexte angepasst werden, um unter allen denkbaren Szenarien noch einen Sinn zu ergeben. Ein gelungenes Beispiel für zusätzliche Labels sehen Sie im folgenden Screenshot aus dem Spendenformular von [sternstunden.de](http://www.sternstunden.de) (Gewinner einer BIENE 2010). Die Labels sind hier durch ihre Betextung in allen der oben genannten Fälle noch so sinnbildend, dass dieses Problem nicht weiter auffällt:



Wie Sie trotzdem mehrere optisch von einander getrennte Elemente mittels eines gemeinsamen Labels verknüpfen können beschreibt der Artikel »Always read the `<label>`« von Richard Northover im Web Developer Blog der BBC. Falls Sie in Ihren Formularen WAI-ARIA einsetzen helfen unter Umständen auch die Attribute `aria-labelledby` und `aria-describedby`, mit denen man eine Verknüpfung auch ohne Label herstellen kann. Näheres dazu im Blog von Marco Zehe: »Easy ARIA tip #2: `aria-labelledby` and `aria-describedby`«

Label verstecken

Ein gern gesehener Fehler ist, dass ein wohlmeinender Frontend-Entwickler aus Accessibility-Gründen ein Label einbaut, wo vom Design her keines vorgesehen ist und dies dann per CSS mit `display:none;` versteckt. Leider kann man sich nicht darauf verlassen, dass Screenreader diese Label noch vorlesen, weil auch sie natürlich vom Browser mit dem fertig verarbeiteten Dokument versorgt werden. Und da heisst `display:none;` übersetzt: »Tu so als wäre dieses Element nicht im Dokumentenbaum vorhanden«. JAWS und NVDA lesen derart versteckte Label zwar vor, wenn das verknüpfte Eingabeelement fokussiert wird, allerdings nur im Virtuellen-Cursor-Modus in der Kombination zusammen mit Internet Explorer, nicht jedoch wenn die Screenreader zusammen mit Firefox eingesetzt werden.

Da man sich also nicht auf diese Form des Versteckens verlassen kann, bietet es sich an, die Labels mit der `offsetLeft`-Methode aus dem Viewport zu schieben – diese wird von Screenreadern wie gewünscht vorgelesen:

```
label {  
  height: 1px;  
  left: -999em;  
  overflow: hidden;  
  position: absolute;  
}
```

Weitere Informationen, wie Screenreader mit versteckten Elementen umgehen, finden Sie im Artikel »Invisible Form Prompts« von Gez Lemon.

Neu! Jetzt mit Knopf zum Selberdrücken!

Auch wenn Labels aus Sicht der Accessibility sicher ein wichtiges Element sind, ohne Abschicken- oder Speichern-Buttons geht gar nichts. Dafür kennt der HTML-Standard gleich zwei Methoden: die verbreitete per `input type="submit"` und das weithin unbeachtete, aber wesentlich flexiblere `BUTTON`-Element. Die Syntax für die Funktionen ist bei beiden Methoden gleich – mit `button type="submit"` wird ein Formular bzw. dessen Inhalt ebenfalls zum Server geschickt.

Werte für Buttons

Bei Elementen ohne Label wie z.B. »Abschicken«-Buttons müssen die Informationen zur ausgelösten Aktion über den jeweiligen Wert des Kontrollelementes (`value="Formular abschicken"`) hinterlegt werden. Wenn Sie diesen Wert nicht setzen, überlassen Sie es damit dem jeweiligen Browser, welcher Text auf dem Button erscheint. Ein einfaches:

```
<input type="submit">
```

ohne Angabe des Attributs `value` ergibt:



Dieser vom Browser nach Gutdünken vergebene Text muss nicht unbedingt dem Ergebnis der ausgelösten Aktion entsprechen. Der Internet Explorer macht z.B. aus dem obigen Code »Anfrage senden«, in Firefox heißt der Text »Daten absenden« (bzw. in älteren Versionen »Anfrage abschicken«), bei Safari, Chrome und Opera ist das Ergebnis ein einfaches »Senden«. Bei den englischen Programmversionen wird vom jeweiligen Browser ein englischer Text eingefügt (»Submit ...«), was in den meisten Fällen wenig wünschenswert ist.

Wenn Sie grafische Buttons nach der Art von `<input type="image" src="button.gif">` beziehungsweise `<button></button>` verwenden, sollten Sie grundsätzlich ein `alt`-Attribut setzen, ansonsten wird der grafische Button von einer Sprachausgabe nicht vorgelesen und das Formular ist damit für Screenreader-Nutzer nicht mehr verwendbar.

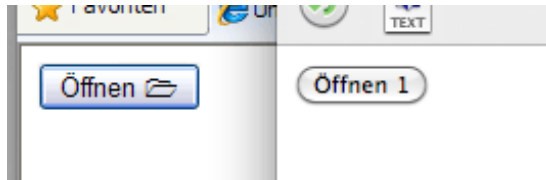
Im Interesse von Nutzern, die Ihre Formulare mit Spracherkennung bedienen, sollte das `alt`-Attribut den exakten Text des grafischen Buttons als Äquivalent enthalten. Die Spracherkennungs-Software orientiert sich am Quelltext und weiss nicht, was in Ihrem grafischen Button steht; der Nutzer wiederum liest den Text des grafischen Buttons. Wenn dieser nicht mit dem maschinenlesbaren Text übereinstimmt, wird die Aktion nicht ausgeführt.

Vermeiden Sie Sonderzeichen

Ein besonderes Problem für viele alternative Zugangsformen ist die gelegentlich zu sehende Verwendung von Sonderzeichen oder so genannten Dingbat-Fonts. Sie können nicht davon ausgehen, dass alle Nutzer die gleichen Zeichensätze wie Sie installiert haben, und insbesondere bei Exoten führt dies oftmals zu unvorhergesehenen Ergebnissen. So zeigt das folgende Codefragment:

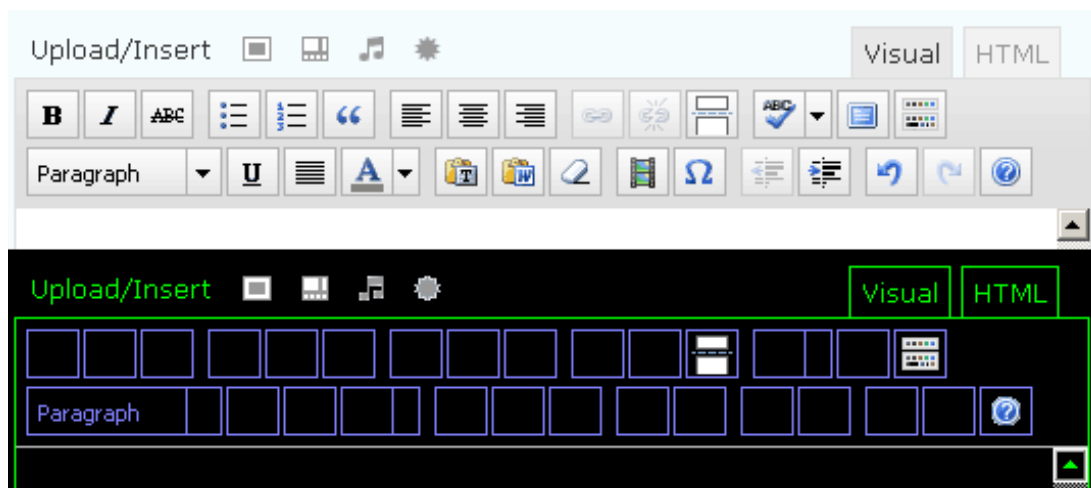
```
<button type="submit">Öffnen<span style="font-family:Wingdings">1</span></button>
```

unter Windows aller Wahrscheinlichkeit nach ein Symbol mit einem sich öffnenden Ordner an, auf anderen Plattformen wird jedoch eventuell die Ziffer »1« dargestellt und auch Screenreader lesen hier ganz konsequent eine Zahl vor.



Vermeiden Sie Hintergrundbilder

Statt der problematischen Sonderzeichen sieht man oftmals auch Icons in Buttons, die als CSS-Hintergrundbilder realisiert sind. Diese Methode der Umsetzungen bringt jedoch ihre eigenen Probleme in punkto Barrierefreiheit mit sich: so sind Hintergrundbilder grundsätzlich nicht mehr sichtbar, sobald der Benutzer eigene Farben z.B. über ein Windows-Kontrastschema eingestellt hat. Fatal sind die Auswirkungen dann, wenn das Hintergrund-Icon der einzige Inhalt z.B. eines Button-Elements ist, insbesondere weil man diesen keinen Alternativtext zuweisen kann



Die Auswirkungen dieser Technik und eine mögliche Lösung per CSS zeigt Steve Faulkner in seinem Artikel »High Contrast Proof CSS Sprites«.

Formulare mit und ohne Tabellen

Mangelnde Linearisierung: Warum Tabellen vollkommen ungeeignet für Formulare sind

Wie beim allgemeinen Layout von Seiten gilt auch für Formulare: Layout-Tabellen sind eine Technik aus dem vergangenen Jahrhundert. Mit Browsern der vierten und fünften Generation war eine durchgängige, also in allen damals vorhandenen Browsern vergleichbare Gestaltung und Funktionalität nur mit Layout-Tabellen erreichbar. Mittlerweile sind diese Browser aber den Gang alles Irdischen gegangen und tauchen in keiner Statistik mehr in Größenordnungen auf, die eine spezielle Berücksichtigung rechtfertigen würden.

Zudem sind die meisten im Netz zu findenden Formulare lediglich eine Abfolge von nacheinander abzuarbeitenden Funktionen, die aber meistens in keiner Beziehung zueinander stehen und somit auch keine tabellarischen Daten darstellen. Layout-Tabellen bringen eine Vielzahl von Problemen mit sich, insbesondere wenn sie nicht in einem der üblichen grafischen Browser ausgegeben werden. Das folgende Beispiel zeigt

eine durchaus gängige Art für die Eingabe persönlicher Daten für den Benutzer in einer unsichtbaren Layout-Tabelle:

Vorname:	Nachname:
<input type="text"/>	<input type="text"/>

Zur Verdeutlichung hier nochmals die gleiche Tabelle, diesmal mit hervorgehobenen Rändern, um die Reihenfolge der Tabellenzellen zu verdeutlichen:

Vorname:	Nachname:
<input type="text"/>	<input type="text"/>

Werden die Inhalte dieser Tabelle nun in linearisierter Form dargestellt, also in der Reihenfolge, in der sie im Quelltext erscheinen, so ergibt sich in einem Screenreader folgender Output:

Vorname Name Textfeld Textfeld

Wenn in einem Formular viele Daten auf die gleiche Art abgefragt werden, verliert der Benutzer einer Sprachausgabe sehr schnell die Orientierung. Er kann die Beschriftungen zu den Formularelementen nicht mehr eindeutig zuordnen. Zwar können Sie die Beschriftungen per Label den jeweiligen Textfeldern zumindest programmatisch zuordnen, die Reihenfolge im Quelltext bleibt jedoch nach wie vor die Falsche und wird unter Umständen auch genau so falsch vorgelesen. Auch eine Zuordnung der Tabellenzellen über die für Datentabellen gedachten Elemente wie TH, scope, axis etc. ist nicht zu empfehlen, da solche Strukturelemente vom Screenreader im Formularmodus ignoriert werden.

Hier muss also eine Lösung gefunden werden, die in den gebräuchlichen grafischen Browsern ein adäquates Layout ermöglicht und trotzdem nicht die Bedienung der Formulare mit alternativen Zugangsmethoden verhindert.

Wenn Formulare mit tabellarischen Darstellungen vermischt werden, dann gelten auch hier die gleichen Regeln für den technischen Aufbau wie zu Beispiel die zwingende Verwendung von Labels. Im folgenden Screenshot aus der erweiterten Verkehrsmittelwahl bei bahn.de ist die Funktion der Checkboxes ohne die Zuordnung per Label nicht ersichtlich, da die Icons für ICE, IC, S-Bahn etc. nicht mit diesen verknüpft sind und die Checkboxes auch über keine sonstigen sprechenden Bezeichnungen (z.B. per title) verfügen:

Verkehrsmittel								
Müngersdorf Alter Militärring, Köln - Mönchengladbach Hbf	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Umstiege zulassen

```

HTML CSS Skript DOM Netzwerk YSlow
tr#productsHeadline < tbody < table#pr...ed.query < div.queryBox < fieldset < form < div.hafasContent < div#hafasContainer
  <td style="width: 22px; text-align: center;">
    <td style="width: 22px; text-align: center;">
       &nbsp;</td>
  </tr>
  <tr id="productRow_0_0" class="">
    <th style="white-space: normal;">
      <td style="text-align: center;">
        <input class="checkbox" type="checkbox" checked="checked" value="1" name="REQ@JourneyProduct_prod_
      </td>

```

Logische Verknüpfungen: Warum Tabellen wunderbar geeignet für Formulare sind

Anders gelagert ist der Fall unter Umständen bei web-basierten Anwendungen. Ein Beispiel: tabellarische Übersichten in Content Management-Systemen sind natürlich mit Tabellen zu strukturieren, da es sich um tabellarische Daten handelt – wenn auch als Mischform mit Formularen z.B. für die eigentlichen

Editierfunktionen. Eine ganz einfache Regel: wenn man die Inhalte anhand der Kriterien in den Spalten- oder Zeilenköpfen sortieren kann, dann ist eine Datentabelle angebracht.

Auch browser-basierte Tabellenkalkulationen wie Google Spreadsheets oder jede andere Form der Eingabe und Manipulation komplexer Daten sind genau das: Tabellen. Hierfür sind die Tabellenelemente aus HTML nach wie vor hervorragend geeignet und somit die nahe liegende Technik zu ihrer Umsetzung. Allerdings gilt auch hier ganz besonders der Zwang zur Linearisierbarkeit, wenn diese Tabellen- und Formulargemische auch in nicht-visuellen Ausgabeformen nutzbar bleiben sollen. Wenn Sie die Inhalte von ihrem Codegerüst befreien, dann müssen sie in genau dieser Reihenfolge immer noch einen Sinn ergeben, denn genau so kommen sie aus dem Lautsprecher.

Allerdings helfen hier die Elemente und Attribute zur Strukturierung, die zwar in Layout-Tabellen unerwünscht sind, in komplexen oder mehrdimensionalen Datentabellen jedoch eine effektive Nutzung erst ermöglichen. Wie diese einzusetzen sind, können Sie in unserer BITV-Reloaded-Serie unter Anforderung 5 sowie im Artikel zu barrierefreien Datentabellen nachlesen.

Was Sie nicht tun sollten ist, die Optik einer Tabelle mit anderen Mitteln wie generischen DIV oder ähnlichem nachzuahmen. Dadurch würden gerade Nutzern nicht-visueller Zugangsformen wichtige Informationen und Anhaltspunkte zur Orientierung in den Datensätzen verloren gehen. Dem Entwickler des im Screenshot zu sehenden Bundesliga-Tabellenrechners hätte eigentlich spätestens beim Wort »Tabelle« auffallen müssen, dass es sich hier um tabellarische Daten handelt und folgerichtig eine HTML-Tabelle die richtigen Elemente zur Verfügung gestellt hätte. Schließlich heisst es ja Bundesliga-Tabelle und nicht Bundesliga-DIV:

BUNDESLIGA-TABELLENRECHNER									
Spieltag 10.		1. FC Köln		3 : 2		HSV			
Meine Tipps					Aktuelle Bundesliga-Ergebnisse				
29.10 20:30	FC Bayern	4:2	Freiburg	Bericht					
30.10 15:30	St. Pauli	1:3	Eintracht	Bericht					
30.10 15:30	1. FCK	3:0	M'gladbach	Bericht					
30.10 15:30	1. FC Köln	3:2	HSV	Bericht					
30.10 15:30	Wolfsburg	2:0	Stuttgart	Bericht					
30.10 15:30	Werder	2:3	Nürnberg	Bericht					
30.10 18:30	Schalke 04	0:1	Bayer 04	Bericht					
31.10 15:30	Mainz 05	0:2	Dortmund	Bericht					
31.10 17:30	Hoffenheim	4:0	Hannover	Bericht					
Meine Tabelle					Aktuelle Bundesliga-Tabelle				
Platz	Team	Sp.	Pkt.	Diff.	Tore				
1	1. FC Köln	10	25	10	19:9				
2	Dortmund	10	22	14	23:9				
3	Mainz 05	10	21	6	17:11				
4	Hoffenheim	10	18	8	21:13				
5	Bayer 04	10	18	4	19:15				
6	Eintracht	10	16	7	17:10				
7	HSV	10	15	1	15:14				
8	Nürnberg	10	15	0	14:14				
9	Freiburg	10	15	-2	16:18				
10	FC Bayern	10	14	1	12:11				

Nur unter Aufsicht anwenden: Strittige Punkte

Es gibt einige Punkte, bei denen selbst unter Experten und Betroffenen keine Einigkeit herrscht, ob sie sinnvoll sind oder ob man diese Techniken auslassen kann und trotzdem noch ein barrierefreies Angebot hat. Ein übermäßiger Einsatz aller in HTML vorgesehenen Hilfen zur Orientierung und Navigation kann sogar bei manchen Nutzergruppen dazu führen, dass eine Seite weniger benutzbar wird. Das ist insbesondere dann der Fall, wenn die eingesetzte Technik einen übermäßigen Lernaufwand erfordert, der die Vorteile der Technik aufwiegt oder sogar übersteigt. Ein solcher Kandidat sind die von den Richtlinien und Verordnungen für Barrierefreie Internetauftritte verlangten Tastaturkürzel zur Navigation, die sogenannten Accesskeys.

Accesskeys – Ja oder Nein?

Die Idee hinter dem auch in HTML5 nach wie vor spezifizierten `accesskey`-Attribut ist eigentlich bestechend einfach: Wie in den gewohnten Desktop-Anwendungen kann man auch in HTML-Seiten Tastaturkürzel definieren, durch die bestimmte Aktionen ausgelöst werden. In HTML können diese Attribute einer Vielzahl von Elementen zugewiesen werden und somit unterschiedlichste Aktionen auslösen. Wie alle Attribute hat

auch dieses einen Wert, der in Form von einzelnen Buchstaben oder Zahlen angegeben wird. Dieses Zeichen entspricht der Taste, die der Benutzer zusammen mit einer oder mehreren weiteren Tasten drücken muss, um die gewünschte Aktion auszulösen. Und genau da beginnt das Problem:

Unter den Browserherstellern herrscht keine Einigkeit, welche Tasten zusätzlich neben dem im Accesskey angegebenen Zeichen zu drücken sind.

- Unter Windows sind dies im Internet Explorer die Alt-Taste + Accesskey + Enter
- Im Mozilla Alt-Taste + Accesskey ohne Enter
- Unter Opera sind es Shift + Esc + Accesskey ohne Enter
- Auf anderen Plattformen wie dem Apple Macintosh sind es statt der Alt-Taste die Ctrl-Taste auch ohne Enter
- Opera macht hier wieder die Ausnahme mit Shift + Esc und so weiter.

Sie sehen die Unmöglichkeit, einen erklärenden Hilfe-Text für eine Website zu verfassen. Zudem sind Accesskeys gerade für viele Menschen mit motorischen Behinderungen, für die sie auch gedacht waren, durch die Notwendigkeit zum gleichzeitigen Drücken mehrerer Tasten kaum zu gebrauchen.

Erschwerend kommt hinzu, dass die meisten Tasten, die man definieren kann, schon in anderen Programmen vergeben sind. Wenn man die gängigsten Browser untersucht und verbreitete assistierende Werkzeuge hinzunimmt, wird man sehr schnell feststellen, dass fast alle Zeichen der Tastatur schon von einem Programm benutzt werden.

Noch schwieriger wird die Situation bei mehrfach vergebenen Werten – dies ist zwar streng genommen ein Fehler, der aber von keinem Validator bemerkt wird und daher in der Praxis öfter vorkommt. Auch hier können sich die verschiedenen Browser nicht einigen, was zu tun ist, wenn ein Nutzer einen Accesskey auslöst, der mehrfach vorhanden ist:

- Firefox geht alle Elemente, denen der gleiche accesskey zugewiesen wurde, in der Reihenfolge ihres Auftretens durch. Aktivierbare Elemente (wie Links und Buttons) werden zwar fokussiert, aber nicht ausgelöst.
- Internet Explorer geht, wie Firefox, alle Elemente, denen der gleiche accesskey zugewiesen wurde, in der Reihenfolge ihres Auftretens durch. Allerdings werden Buttons sofort ausgelöst, d.h. Daten unter Umständen übergeben, die der Nutzer noch gar nicht absenden wollte.
- Webkit (Safari/Chrome) löst das letzte Element aus, dem einer dieser mehrfach vergebenen Accesskeys zugewiesen ist.
- Opera löst das erste Element aus, dem einer dieser mehrfach vergebenen Accesskeys zugewiesen ist.

Weitere Hintergründe zu diesem Thema finden Sie im Artikel »If you use the accesskey attribute, specify unique values« von Roger Johansson.

Noch problematischer ist, dass jeder Webentwickler entscheiden kann, wie er diese Tasten vergibt. Für die Vergabe der Accesskeys gibt es keine einheitliche Richtlinie, so dass der Benutzer auf jeder Website, die er neu ansurft, zunächst einmal die Accesskeys erlernen müsste. Diese nicht kontrollierbaren Folgen sind einer der Gründe, warum z.B. die kanadische Bundesregierung die Verwendung von Accesskeys wieder aus ihrem Style Guide gestrichen hat.

Ausnahme: Wenn Sie eine web-basierte Anwendung mit vielen ähnlichen Formularen für ein Intranet entwickeln, können Sie natürlich genauere Vorhersagen darüber treffen, welche Accesskeys in Ihrer Umgebung funktionieren. In diesem Fall haben Sie ja die volle Kontrolle über die zu verwendende Zugangssoftware. Bei einem solchen System kann die Definition von Accesskeys für immer wiederkehrende Funktionen (also z.B. immer das erste Eingabefeld oder unterschiedliche logische Blöcke) sinnvoll und lohnenswert sein. In diesem Fall reden wir aber von Webinhalten, die keinen Dokumenten-Charakter haben, sondern eindeutig Anwendungs-Charakter. Bei Webseiten, die lediglich aus strukturierten Texten, der dazugehörigen Navigation und gelegentlichen simplen Formularen bestehen, lohnt sich der Einsatz generell nicht.

Tabindex – Ja oder Nein?

Ein enger Verwandter des `accesskey`-Attributes ist der `tabindex`. Mit diesem Attribut können Sie die Reihenfolge festlegen, in der sich ein Anwender per Tabulator-Taste durch eine Seite bewegen kann. Dies geschieht jedoch nicht in willkürlicher Form wie bei den Accesskeys, sondern in der Reihenfolge, die Sie bei der Erstellung des Formulars definieren:

```
<input type="text" id="suchfeld" size="20" tabindex="1">
```

Da die meisten Browser und assistierenden Programme von sich aus die Navigation per Tabulator-Taste unterstützen, brauchen Sie nicht jedes Element einer Seite mit einem `Tabindex` zu dekorieren. Dies wäre kontraproduktiv, da Sie nicht vorhersagen können, in welcher Reihenfolge der Benutzer durch die Seite navigieren möchte oder an welcher Stelle der Benutzer mit der Navigation beginnt. Zudem entspricht die Tabreihenfolge grundsätzlich dem Ablauf des Quellcodes – wenn sie hier für eine vernünftige und nachvollziehbare Abfolge sorgen, dann ist das `tabindex`-Attribut nicht mehr nötig.

Sinnvoll ist der Einsatz des `tabindex`-Attributes in Formularen, wenn die Reihenfolge, in der Kontrollelemente bedient oder Texteingaben gemacht werden sollten, nicht der Reihenfolge dieser Elemente im Quelltext entspricht. Dies kann zum Beispiel bei der Eingabe von Postleitzahl und Ort in zwei unterschiedliche Felder der Fall sein. Diese werden vom Benutzer in genau dieser Reihenfolge erwartet, müssen aber im Quelltext nicht unmittelbar aufeinander folgen und können unter Umständen nicht nacheinander von der Tabulator-Taste angesprungen werden.

Gerade bei mehrspaltigen Formularen muss die optische Anordnung nicht unbedingt den Erwartungen der Nutzer und der Abfolge im Quelltext entsprechen. Da Nutzer in Formularen ja bereits die Tastatur zur Eingabe benutzen müssen, steigt die Wahrscheinlichkeit, dass die Navigation innerhalb des Formulars ebenfalls per Tastatur (Tab-Taste, Pfeiltasten, Enter etc.) stattfindet – eine sinnvolle Tabreihenfolge wird dadurch umso wichtiger.

Wenn Sie in Ihren Formularen `tabindex` setzen wollen, dann sollten Sie diese für die LABEL-Elemente und nicht für die eigentlichen Kontrollelemente (Textfeld, Checkbox usw.) definieren. So stellen Sie sicher, dass diese Beschriftungen von einer Sprachausgabe zugeordnet und vorgelesen werden.

Tipp: Wenn Sie in mehreren Bereichen einer Seite Werte für `tabindex` vergeben, können Sie dies auch in 10er-Schritten machen. Mit diesem kleinen Kniff brauchen Sie bei Erweiterungen mitten in einer Seite nicht alle folgenden Elemente mit `tabindex`-Attribut neu zu numerieren. Uns ist bisher noch kein Browser begegnet, der dieses Attribut zwar versteht, aber nicht von `tabindex="14"` nach `tabindex="20"` springen kann, wenn die Werte dazwischen nicht vergeben sind.

Vorbelegung von Formularfeldern

Ein weiterer Punkt, an dem sich die Geister scheiden: die Vorbelegung von Formularelementen wie etwa Textfeldern durch einen Text, der im jeweiligen Feld angezeigt wird. Die Wurzeln der entsprechenden Vorgaben in den WCAG und damit auch der BITV-Bedingung 10.4 stammen noch aus einer Zeit, als angeblich manche Screenreader leere Textfelder nicht erkannten und darüber hinweg lasen. Uns ist jedoch kein einziges Hilfsmittel bekannt, das nach wie vor diesen Fehler aufweist, somit lässt sich diese Forderung der BITV nicht weiter aufrechterhalten. Weitere Informationen hierzu finden Sie in unserer BITV-Serie unter Bedingung 10.4, Testergebnisse verschiedener Methoden zum Vorbelegen und Löschen finden Sie im Artikel »Testing Accessibility of Pre-populated Input Fields« von Terrill Thompson.

Übermäßiger Einsatz von title-Attributen

Nicht nur in Navigationsleisten, auch in Formularen findet man häufig Inhalte und Funktionen, deren Sinn und Zweck sich erst erschließt, wenn man mit der Maus ein Tooltip auslöst. In diesen per `title`-Attribut realisierten kleinen Fähnchen stehen dann oftmals erläuternde Hinweise dazu, was sich hinter der jeweiligen Funktion verbirgt, da die Funktion selbst nicht verständlich beschriftet ist. Wenn Ihre Funktionen ohne diese Tooltips nicht verständlich sind – sollten Sie dann nicht lieber den Aufwand für die technische Umsetzung und Betextung dieser `title`-Attribute in verständlichere Icons, Buttons und Links stecken? Zumal diese `title`-

Attribute nur bei Mausbedienung erscheinen, für Tastaturnutzer oder auf den Touchscreens von Tablet-PCs oder Smartphones bleiben sie üblicherweise verborgen.

title-Attribute als Label-Ersatz

Berechtigte Ausnahmen von diesem Einwand gibt es aber dennoch: das `title`-Attribut kann als Ersatz für Label genommen werden, wenn diese aus irgendeinem Grund nicht einsetzbar sind (seien es Platzgründe oder aus technischen Gründen, die dagegen sprechen). Screenreader behandeln diese `title` dann wie LABEL und lesen sie entsprechend vor (selbst dann, wenn das Programm so eingestellt ist, dass eigentlich keine `title` vorgelesen werden sollen). Diese Vorgehensweise ist im Übrigen auch durch die Richtlinien der WCAG abgedeckt, näheres dazu in den Techniken unter »H65: Benutzung des title-Attributs, um Formular-Steuerelemente zu kennzeichnen, wenn das Label-Element nicht benutzt werden kann«.

Diese Eigenheit von Screenreadern können Sie ausnutzen, um ganz gezielt Informationen an die Nutzer solcher Hilfsmittel zu kommunizieren (Bildschirm lupen haben ebenfalls vergleichbare Funktionen, um die Inhalte von `title`-Attributen anzuzeigen, wenn ein Element den Fokus erhält). Ein Beispiel für den sinnvollen Einsatz wären Funktionen, die durch ihr Layout oder andere optische Eigenschaften verdeutlicht werden und man befürchten muss, dass diese Information in der Sprachausgabe, bei extremer Vergrößerung oder bei benutzerdefinierten Farben verloren gehen.

Weitere Gedanken zum Einsatz von `title`-Attributen finden Sie im Artikel »Using titles on form fields« von Mike Davies.

HTML5 jenseits des Hypes

Ein wenig Geschichte

Die konzeptionellen Vorarbeiten für HTML5 begannen irgendwann in den Jahren 2003/2004 bei Opera unter dem Namen Web Applications 1.0. Bald wurde die Sprache von einem Zusammenschluss verschiedener Browser-Hersteller jenseits der Standardisierungs-Gremien des W3C weiterentwickelt. Diese so genannte *Web Hypertext Application Technology Working Group* (WHATWG) wurde von Apple, Mozilla und Opera getragen und hatte zum Ziel, die mittlerweile seit über 10 Jahren bestehenden Lücken in HTML4/XHTML1 durch die Festlegung neuer Elemente zu schließen, die für zeitgemäße Web-basierte Applikationen dringend benötigt wurden.

Im Jahr 2007 installierte das W3C schließlich eine neue HTML-Arbeitsgruppe (die alte war de facto aufgelöst und XHTML2 hatte sich als Sackgasse erwiesen) und übernahm somit die Weiterentwicklung der Spezifikation. Zurzeit wird mit der Veröffentlichung einer sog. Candidate Recommendation für das Jahr 2012 gerechnet.

Sagen was gemeint ist: präzisere Beschreibung durch neue Elemente

Dass der neue Standard noch nicht fertig ist, sollte aber niemanden vom Einsatz der neuen Elemente abhalten – im Gegenteil: die meisten modernen Browser bieten mittlerweile einen hohen Grad der Unterstützung, der zudem laufend verbessert wird, und in der Zwischenzeit ist sogar Microsoft mit an Bord.

HTML5 bringt neben einer ganzen Reihe neuer Elemente, der vereinfachten Syntax und den JavaScript-APIs nun endlich ein weitgehend standardisiertes Browserverhalten. Im Gegensatz zu den älteren Spezifikationen wird hier sehr detailliert beschrieben, was Browser mit den einzelnen Elementen einer HTML5-Seite machen sollen und selbst die möglichen Fehler und wie Browser sich von ihnen »erholen« ist im Detail festgelegt.

Der Nachteil dieser Genauigkeit: die gesamte Dokumentation ist dadurch wesentlich umfangreicher geworden. @thomasfuchs

Weitere Links zum Thema:

- Cristian Colceriu: »How to Build Cross-Browser HTML5 Forms«
- Paul Irish: »HTML5 Boilerplate – A rock-solid default template for HTML5 awesome.«

bemerkte bei Twitter, dass die gesamte HTML-Spezifikation im Jahre 1998 mit 9.967 Worten auskam – mittlerweile hat allein das einleitende HTML5-Overview-Dokument 311.000 Wörter!

Wie immer bei noch nicht endgültig verabschiedeten Standards kann und wird sich noch einiges ändern. Im Extremfall können auch Dinge wieder entfallen wenn sich herausstellt, dass sie in keinem Browser vernünftig umgesetzt werden. Daher sollte man, wenn man in seiner Applikation auf HTML5 setzt, die Fortentwicklung genau beobachten.

Übrigens: Nicht Teil von HTML5 ist CSS3, auch wenn sich manche Artikel im Netz so lesen – dies ist nach wie vor ein eigenständiger Standard.

Was geblieben ist: HTML legt nach wie vor lediglich die Struktur der Inhalte fest. Es tut dies aber auf eine Art und Weise, die eine viel präzisere Beschreibung der Inhalte und Funktionen einer Seite oder einer Applikation erlaubt. In Verbindung mit den neu definierten APIs werden komplexe Interaktionen ermöglicht, die bisher nur mit sehr viel JavaScript oder mit Fremdformaten wie Flash (auf Kosten der Zugänglichkeit) zu erreichen waren.

Dies gilt besonders für den Bereich der Formulare, auch wenn hier die Unterstützung in den verschiedenen Browsern noch sehr uneinheitlich und lückenhaft ist. Bei den neuen HTML5-Elementen zur Festlegung der Seitenstruktur ist der Einbau noch relativ einfach und ohne gravierende Nebenwirkungen zu erreichen. Selbst Browser wie Internet Explorer 6 bis 8 kann man mit einem kleinen Schubs per JavaScript dazu bringen, dass sie diese Elemente verstehen:

```
document.createElement('header');
document.createElement('footer'); ...
```

Andere Browser brauchen hingegen etwas Nachhilfe im CSS, um die neuen Elemente wie gewünscht darzustellen. Da unbekannte Elemente laut Spezifikation von HTML5-konformen Browsern zunächst einmal als inline-Elemente verarbeitet werden müssen, muss man hier noch explizit angeben, dass bestimmte Elemente als sog. block-level-Elemente gerendert werden sollen. Dies ist jedoch zum Glück mit einigen wenigen Zeilen Code erledigt:

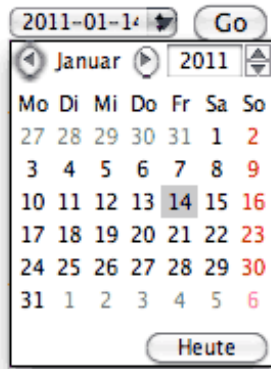
```
header, footer, ...{
  display: block;
}
```

Bei dem guten Dutzend neuer Formular-Elementen ist dies leider nicht ganz so einfach. Zwar haben die neuen Input-Elemente den nicht zu unterschätzenden Vorteil, dass in allen relevanten älteren Browsern an ihrer Stelle ein althergebrachtes Textfeld (`<input type="text">`) dargestellt wird, wenn die neuen Datentypen nicht erkannt werden. Allerdings braucht man für solche Fälle dann immer noch ein Fallback, um den Nutzern solcher Browser zum Beispiel die Auswahl eines Datums aus einem klassischen, JavaScript-basierten Kalender-Widget zu ermöglichen.

Auch in aktuellen Browsern kann man nicht davon ausgehen, dass diese z.B. ein natives Kalender-Widget (`<input type="datetime">`) auch anzeigen, wenn dies vom Autor des Formulars so gewünscht wird – auch hier sind unter Umständen Fallback-Lösungen nötig, zumal es in vielen Browsern auch mit der Barrierefreiheit dieser Widgets nicht so weit her ist. So sind zum Beispiel viele der Elemente selbst in manchen modernen Browsern schlichtweg nicht per Tastatur bedienbar.

Hinzu kommt das Problem, dass diese Widgets generell nicht per CSS formatierbar sind, weil sie nicht Teil des DOMs sind. Man übergibt also die volle Kontrolle über Darstellung und Bedienung an den Browser – mit teils ungeahnten Konsequenzen: der nebenstehende Screenshot bereitete uns einige Mühe, weil Opera bei jedem Versuch die Schrift zu skalieren, reproduzierbar abstürzte.

- Elco Klingen: »HTML5 elements in Internet Explorer without Javascript«
- Splashnology: »Useful HTML5 & CSS3 Toolbox For Web Developers«
- Chris Spooner: »Create a Stylish Contact Form with HTML5 & CSS3«
- Devlounge: »HTML5 Forms Styled With CSS3«
- Vlad Alexander: »The shortcomings of HTML5«
- Bruce Lawson: »HTML 5 forms Demo«



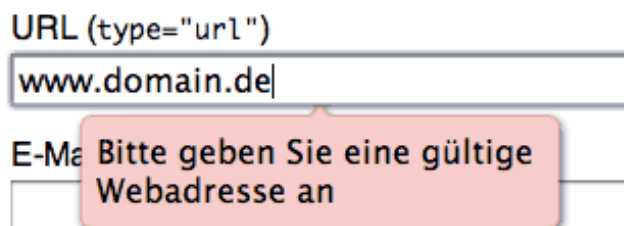
Was gibt's Neues?

Zunächst einmal eine ganze Reihe neuer Input-Typen und zusätzliche Attribute wie zum Beispiel:

- `<input type="email">`
- `<input type="date">`
- `<input type="time">`
- `<input type="month">`
- `<input type="week">`
- `<input type="datetime" ...>`
- `<input type="range">`
- `<input type="number">`
- `<input type="color">`
- `<input type="search">`
- `<input type="file" multiple>`
- `<input... autofocus>`
- `<input... autocomplete>`
- `<input... required>`
- `<input... pattern="[a-z]{3}[0-9]{3}">`

(Die althergebrachten Input-Typen wie z.B. `button`, `file`, `hidden`, `image`, `password`, `reset`, `submit` und `text` sind natürlich weiterhin Teil des HTML-Standards.)

Ein immenser Vorteil dieser Input-Typen ist, dass sie von sich aus nur bestimmte Eingaben oder Auswahlen ermöglichen, ohne dass man die dahinter liegende Logik skripten muss – dies erledigt ein geeigneter Browser von ganz allein. So werden Browser in die Lage versetzt, bei bestimmten Typen die Eingaben direkt zu validieren, ohne dass dafür JavaScript ausgeführt oder der Server konsultiert werden muss. Hier ein Beispiel für Input-Felder vom Typ `"url"` in Opera 11:



Bisher musste man solche Eingaben client- und serverseitig mit regulären Ausdrücken auf ihre Gültigkeit überprüfen. Nun lassen sich über einfache Attribute nicht nur Pflichtfelder festlegen (`required`-Attribut), sondern der Browser prüft auch gleich noch, ob es sich bei der Eingabe um einen der korrekten Syntax entsprechenden URL handelt.

```
<form action="foo.bar" method="get">
  <label for="url">URL:</label>
  <input type="url" id="url" name="url" required>
  ...
</form>
```

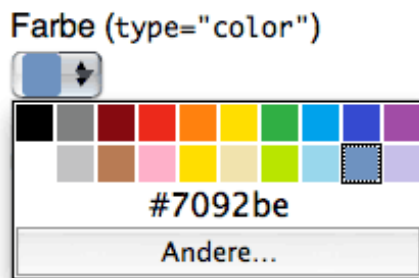
Mit Hilfe des `pattern`-Attributs lässt sich dann noch überprüfen, ob z.B. bestimmte Vorgaben an das Format (z.B. `https://` für eine gesicherte Verbindung) eingehalten wurden.

Noch ein paar konkrete Beispiele

(Eine Übersicht der neuen Formularelemente finden Sie in der Beispieldatei `html5forms.html`)

`<input type="color">`

Feld für Farbauswahl.



Hier die Darstellung in Opera 11, andere Browser stellen unter Umständen ein anders strukturiertes Eingabefeld dar, in das hexadezimale RGB-Farbangaben (0–9, A–F) eingegeben werden können.

`<input type="date">`

Feld für Datumsangabe.

`<input type="datetime-local">`

Feld für Datum und Uhrzeit (ohne Zeitzone).

`<input type="datetime">`

Feld für Datum und Uhrzeit (mit Zeitzone).



(Darstellung in Opera 11).

`<input type="month">`

Tipp: falls Sie die browserseitige Validierung verhindern wollen und sich lieber auf ihre eigenen Methoden verlassen möchten, können Sie die `novalidate`- und `formnovalidate`-Attribute setzen, die dies effektiv verhindern.

Feld für die Angabe von Monat und Jahr.

<input type="number">

Feld für eine Zahl, ggf. mit Pfeiltasten zur Veränderung der Werte (sog. numeric stepper).

Zahl (type="number")



<input type="range">

Feld für eine Zahl, die in einem bestimmten Wertebereich liegen muss.

Bereich (type="range")



(Wird üblicherweise als Schieberegler gerendert – hier die Darstellung in Safari 5)

Dieser Input-Typ akzeptiert weitere Attribute zur Festlegung des gewünschten Bereichs, aus dem der Nutzer etwas auswählen soll ("max" für den Maximalwert, "min" für den Minimalwert, "step" für die Anzahl der Schritte und "value" für den eingestellten Wert):

```
<form action="foo" method="post">
  <input type="range" id="range" min="0" max="10" step="1" value="">
</form>
```

Ohne weitere Angaben wird sich hier dem Benutzer jedoch nicht erschließen, was von ihm erwartet wird. Die aktuellen Browser sind hier auch keine große Hilfe, weil sie mögliche Werte und aktuelle Zustände nicht von sich aus ausgeben. Hier hilft ein wenig CSS, mit dem man die min- und max-Werte auslesen und jeweils vor bzw. hinter den Schieberegler schreibt:

```
input[type=range]:before { content: attr(min); }
input[type=range]:after { content: attr(max); }
```

Zum auslesen des aktuell eingestellten Wertes wiederum braucht man auch mit HTML5 noch etwas JavaScript. Zur Darstellung des ausgelesenen Wertes eignet sich hervorragend das neue Output-Element:

```
<output id="ergebnis"></output>
```

<input type="search">

Feld für Suchbegriffe.

Search (search)



(Darstellung in Safari 5).

<input type="tel">

Feld für Telefonnummern.

Hier sieht man einen weiteren, nicht zu unterschätzenden Vorteil spezialisierter Elemente für bestimmte Datentypen: Browser können je nach Typ verschiedene Eingabearten präsentieren. So zeigt der mobile Safari auf dem iPhone bei einem Input vom Typ "tel" statt der Tastatur den wesentlich passenderen Ziffernblock zur Eingabe einer Telefonnummer an:

(Darstellung in Mobile Safari unter iOS).

`<input type="time">`

Feld für Uhrzeit.

(Darstellung in Opera 11).

`<input type="url">`

Feld für URLs.

Auch hier zeigt sich wieder die ganze Eleganz der neuen Elemente und Ihr Nutzen im Sinne der Anwender: Browser, die diese Elemente verstehen, können dann ein geändertes Interface zur Bedienung präsentieren und damit den Nutzer unterstützen. So zeigen die mobilen Versionen von Safari (iPad, iPhone, iPodTouch) ein leicht geändertes Keyboard beim Ausfüllen von `input type="url"`, bei dem zum Beispiel die Leerschritt-Taste entfällt (da Leerschritte in URLs nicht zulässig sind); stattdessen werden für URLs benötigte Tasten wie `"/` oder `".com"` angezeigt:

`<input type="week">`

Feld für Angabe von Kalenderwoche und Jahr.

`<input type="email">`

Feld für E-Mail-Adressen.

E-Mail (type="email")

Bitte geben Sie eine gültige E-Mail-Adresse an

```
<form action="foo.bar" method="get">
  <label for="email">E-Mail:</label>
  <input type="email" id="email">
  <button type="submit">Abschicken</button>
</form>
```

Wenn Sie ein Feld derart auszeichnen, weisen Sie damit den Browser an, nur bestimmte Eingaben zu akzeptieren, die in ihrer Syntax einer gültigen E-Mail-Adresse entsprechen, z.B. mindestens ein beliebiges Zeichen vor einem @, mindestens zwei Zeichen (inkl. Zahlen und Umlauten) nach dem @, gefolgt von einem Punkt, wiederum gefolgt von mindestens zwei beliebigen Buchstaben). Browser, die diesen und andere Input-Typen nicht verstehen, stellen eine herkömmliche Textbox dar – allerdings auch ohne die eingebaute Validierung, so dass Sie weiterhin eine client- und / oder serverseitige Überprüfung benötigen. Wie für alle neuen Input-Typen gilt also: Sie können sie ohne Nachteile verwenden, aber sie können sich nicht zu 100% darauf verlassen.

<input _ autocomplete="on | off">

Beeinflusst die automatische Vervollständigung eines input-Feldes.

Manchmal kann es sinnvoll sein, das automatische Ausfüllen bzw. Komplettieren von Nutzereingaben zu verhindern, z.B. wenn Angaben nicht aus dem Adressbuch des Nutzers übernommen werden sollen, oder um das automatische Ausfüllen von Benutzernamen und Passwörtern, Kreditkartennummern etc. zu verhindern (autocomplete="off"). Beachten Sie dass Eingaben in solche Felder von HTML5-konformen Browsern gelöscht werden, wenn der Benutzer den »Back«-Button bedient: »... if the resulting autocompletion state is off, values are reset when traversing the history«.

<input pattern="...">

Attribut für Eingaben, auf die ein regulärer Ausdruck passen muss.

Durch das pattern-Attribut lassen sich solche regulären Ausdrücke direkt ins Markup des betreffenden Elementes schreiben:

```
<form action="" method="post">
  <label for="username">Create a Username: </label>
  <input type="text"
    id="username"
    placeholder="8 < 20"
    pattern="[A-Za-z]{4,10}"
    required
    autofocus>
  <button type="submit">OK</button>
</form>
```

[a-zA-Z] bedeutet, dass das Feld nur Klein- und Großbuchstaben akzeptiert, {8,20} bedeutet, dass es mindestens 8 und höchstens 20 Buchstaben sein dürfen. Empfehlenswert ist dies zum Beispiel auch bei Telefonnummern – die Angabe von pattern="\+[0-9\-\-]" ermöglicht die Eingabe von Telefonnummern nach der Art +492282092-0.

<input placeholder="...">

Attribut für Felder mit Platzhalter-Texten, die Hinweise auf die einzugebenden Daten geben sollen.

Vor HTML5 musste man JavaScript einsetzen, um in einem Textfeld eine Vorbelegung mit mehr oder minder sinnvollem Hinweistext zu erreichen, dann prüfen ob der Nutzer das Feld fokussiert hat und den

Platzhaltertext entfernen, dann erneut testen ob der Benutzer etwas eingegeben hat wenn er das Feld wieder verlässt, und gegebenenfalls den Platzhaltertext wieder einfügen (ohne aber bereits gemachte Eingaben wieder zu überschreiben).

Diesen ganzen Aufwand können Sie sich sparen, wenn Sie das placeholder-Attribut aus HTML5 einsetzen – dann macht der Browser diese Arbeit nämlich von ganz alleine:

```
<input type="email" id="email" placeholder="name@domain.de">
```

E-Mail (type="email" + placeholder)



Die Unterstützung hierfür ist nach unseren Tests allerdings noch lückenhaft. Erst die neuesten Versionen von Firefox 4, Opera 11 und Safari 5 unterstützen dieses Attribut – wenn Sie einen hohen Prozentsatz von Nutzern mit älteren Browsern haben und essentielle Hinweise in diesem Platzhalter stehen haben sollten, dann brauchen Sie auch hier einen Fallback.

Zum Glück gibt es eine ganze Reihe Skripte wie zum Beispiel Modernizr, um die Unterstützung von placeholder und anderen Attributen zu testen und entsprechend zu reagieren.

Was Sie nicht machen sollten ist, das placeholder-Attribut als LABEL-Ersatz zu missbrauchen – diese werden nach wie vor benötigt, weswegen dies in der HTML5-Spezifikation auch ausdrücklich verboten ist. Längere Hinweistexte sind nach wie vor besser in einem title-Attribut oder noch besser im sichtbaren Anleitungstext des Formulars aufgehoben. Tests zur barrierefreien Umsetzung dieses Attributs finden Sie unter html5accessibility.com: »HTML5 placeholder attribute tests«.

<input _ required>

Attribut zur Kennzeichnung von Pflichtfeldern.

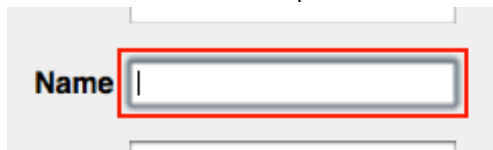
Wie der Name impliziert können Sie hiermit Pflichtfelder auszeichnen. Dies geht entweder, indem Sie das Attribut ohne Wert einfügen:

```
<input type="text" id="foo" required> ,
```

oder wenn Sie die Schreibweise von XHTML bevorzugen auch als Attribut-Wert-Paar:

```
<input type="text" id="foo" required="required" />
```

Funktional sind diese beiden Schreibweisen identisch: mit diesem Attribut kann ein Formular nicht abgesendet werden, wenn das Input-Feld leer ist. Falls der Nutzer hier nichts eingetippt hat wird das Textfeld vom Browser entsprechend markiert.



(Darstellung in Firefox 4)

Das befreit Sie allerdings nicht von der Notwendigkeit, Nutzer auch schon vor dem Ausfüllen und Absenden auf das Vorhandensein von Pflichtfeldern hinzuweisen. Für halbwegs moderne Browser könnten Sie dies theoretisch per CSS erledigen:

```
input[required]:after {  
  content: " * ";  
}
```

Vom Standpunkt der Barrierefreiheit ist diese Variante aber eher ungeeignet, weil z.B. im Screenreader diese essentielle Information fehlt, wenn sie ausschließlich via CSS (und damit über die visuelle Präsentationsschicht) vermittelt wird. Zudem werden solche Attribut-Selektoren erst von der neuesten Browser-Generation unterstützt. Auch hier sollte man weiterhin auf die klassische Kennzeichnung mit Symbolen vertrauen, ggf. ergänzt durch ARIA-Attribute wie `aria-required`.

<input _ autofocus>

Attribut zur automatischen Fokussierung des Kontrollelements.

Auch hier wird durch Neuerungen in HTML5 die Notwendigkeit zum Einsatz von JavaScript reduziert; aber auch hier kann es bei unbedachtem Einsatz zu Problemen in der Zugänglichkeit der Anwendung führen. Das Problem entsteht hier nicht durch HTML5, sondern generell durch das Konzept, ein Formularfeld automatisch zu fokussieren (vergleichbares ließ sich bisher ja auch durch JavaScript erreichen):

```
<input type="text" id="foo" autofocus>
```

Wie Bruce Lawson in seinem Artikel »The accessibility of HTML 5 autofocus« zu Recht bemerkt gibt es nur sehr wenige legitime Einsatzzwecke für diese Technik. Nur wenn sie mit absoluter Sicherheit davon ausgehen können, dass der Zweck des Besuchs der Seite ausschließlich das Erreichen des entsprechenden Formularelementes ist, sollten Sie dieses auch automatisch fokussieren.

Dieses Szenario mag auf Seiten wie die Google-Suchseite zutreffen – diese erfüllt nur den einen Zweck, nämlich dass Nutzer einen Suchbegriff in das Suchfeld eintippen. Hier lässt sich ein autofocus also durchaus rechtfertigen: Sie können davon ausgehen, dass der Nutzer genau deswegen auf die Seite gekommen ist und Sie helfen ihm damit, seine Aufgabe schneller zu erledigen.

Eher hinderlich ist dieses Verhalten auf Seiten deren primärer Zweck eben nicht die Bedienung eines Formulars ist, sondern zum Beispiel das Lesen eines Artikels – ein Fehler den viele Templates des verbreiteten Blog-Systems Wordpress machen. Nutzer von Screenreadern finden sich dort mitten in einem Formular wieder, das Programm schaltet in den Formularmodus und dem Nutzer fehlen jegliche Informationen zum Kontext des Formulars oder zum eigentlichen Inhalt der Seite.

Formulare ohne Form

Eine der spannendsten Neuerungen in HTML5 ist, dass Formularelemente nun auch ausserhalb des <form>...</form>-Tags stehen dürfen. Input muss nun also nicht mehr zwingend ein Kind-Element von FORM sein. Damit entfällt der Zwang, den ganzen BODY einer Seite in ein <form>...</form>-Tag zu packen, wenn Formular-Elemente an verschiedenen, weit auseinander liegenden Stellen der Seite vorkommen (z.B. eine einfache und eine erweiterte Suche, die beide auf die gleiche form action="..." auf dem Server zurückgreifen). Alles was Sie benötigen ist eine eindeutige ID am Start-Tag von <form>:

```
<form id="suche" action="suche.pl" method="post">
  <fieldset>
    <legend>Erweiterte Suchoptionen</legend>
    <input type="search">
    ...
  </fieldset>
</form>
```

Ein weiterer Suchschlitz kann nun irgendwo auf der Seite ausserhalb von <form>...</form> platziert werden; die Verbindung zum Suchskript wird über den Verweis auf die ID des <form>-Tags hergestellt. Dieses verhält sich durch das zusätzliche form-Attribut nun als wäre es Teil des Formulars, auf das der Wert des Attributs verweist:

```
<input type="search" name="suchfeld" form="suche">
```

In der Zwischenzeit: WAI-ARIA

Die grossen Betriebssysteme mit ihren grafischen Benutzeroberflächen verfügen alle über sogenannte Accessibility-APIs (*Application Programming Interfaces*). Über diese Schnittstellen werden Hilfsmittel wie Braillezeilen, Lupenprogramme und Screenreader, aber auch alternative Eingabegeräte mit den nötigen Informationen darüber versorgt, was gerade auf dem Bildschirm geschieht und wie Funktionen zu bedienen sind.

Für Windows ist dies die mittlerweile recht betagte »Microsoft Active Accessibility«-Schnittstelle (MSAA) bzw. für aktuelle Windows-Versionen UI Automation; unter Linux gibt es das »Gnome Accessibility Toolkit« (ATK);

bei Apple ist es das »Accessibility-API for Cocoa«; auch Sun bzw. Oracle hat mit dem »Java Accessibility-API« eine entsprechende Schnittstelle. Um über diese Schnittstellen kommunizieren zu können und dem Nutzer den Zugriff zu gewährleisten brauchen Elemente einer Webseite bzw. eines Web-Formulars klar festgelegte Rollen und Zustände (Roles = Was ist dieses Element und was kann ich damit machen? States = In welchem Zustand befindet sich das Element?)

Im Falle statischer HTML-Dokumente mit den üblichen Strukturelementen wie Überschriften, Absätze und Listen, Links und einfachen Formularelementen sind die Rollenverteilungen und Zustände klar verteilt und sollten somit kein Problem für die Hilfsmittel darstellen. Der Browser teilt die Informationen über die jeweiligen Elemente und ihre Zustände (fokussiert, gedrückt, angekreuzt, besucht) über das Accessibility-API mit.

Wenn nun jedoch per HTML, CSS & JavaScript Bedienelemente nachgebildet werden, die über den begrenzten Satz von HTML4 bzw. XHTML1 hinausgehen (gemeinhin also sinnfreie DIVs oder Bilder sind), eröffnet sich das Problem, dass diese »Nachbauten« von der Accessibility-Schnittstelle nicht verstanden werden und somit auch nicht an ein Hilfsmittel weitergereicht werden können. Screenreader können also gar nicht umhin, solche eigentlich interaktiven Elemente wie statischen Text zu behandeln.

Falls Ihre Anwendung diese Elemente bzw. deren Funktionalität benötigt, so können Sie für moderne Browser die neuen Elemente aus HTML5 auch heute schon verwenden. Allerdings sollten Sie sich dann im Klaren sein, dass nicht ganz so moderne Browser und ältere Hilfsmittel zusätzliche Unterstützung in Form von Fallback-Lösungen und/oder JavaScript benötigen, um die gewünschten Funktionen zur Verfügung zu stellen.

ARIA – Was ist das?

Dieser Spagat wird nun durch den Einsatz der noch in der Entwicklung befindlichen Spezifikation WAI-ARIA erleichtert, die als eine Art Brücken-Technik die Kluft zwischen der begrenzten Welt von HTML4 und den erweiterten Möglichkeiten von HTML5 zu überbrücken sucht. WAI-ARIA ist genau dafür gedacht: Hilfsmittel wie Screenreader und Lupenprogramme mit Informationen über Rollenverteilungen und Zustände von Objekten zu versorgen, an die diese aus eigener Kraft nicht kommen könnten.

Dazu werden HTML-Elemente mit zusätzlichen semantischen Informationen über ihre Funktion, über mögliche Werte und Zustände (an/aus, auf/zu, Prozentwerte bei Schieberegler etc.) und über ihr Verhalten ausgestattet. Dies geschieht über Attribute, die zum Start-Tag des jeweiligen Elementes hinzugefügt werden.

Ein einfaches Beispiel hierfür sind die ARIA-Landmark-Roles: hiermit können die heutzutage üblichen, aus einer (Un-)Menge DIVs aufgebauten Layouts mit weiteren Informationen über ihren Zweck und ihre Inhalte ergänzt werden. Die ARIA-Rolle `role="navigation"` sagt dem Hilfsmittel, dass sich innerhalb des derart ausgezeichneten Bereichs die Navigation der Seite befindet, `role="main"` kennzeichnet den wesentlichen Inhalt einer Seite, `role="complementary"` kennzeichnet ergänzende Informationen zu diesem Inhalt und `role="search"` ist ein Such-Widget – gemeinhin Dinge, für die es (noch) keine passenden HTML-Elemente gibt.

Weitere ARIA-Attribute kümmern sich um Zustände und Eigenschaften: `aria-required="true"` sagt z.B. bei einem Eingabefeld, dass dies ein Pflichtfeld ist, `aria-invalid="true"` weist auf eine Fehleingabe hin:

```
<input type="text" aria-required="true" aria-invalid="true">
```

ARIA-Rollen können auch verschachtelt sein, teilweise haben Sie auch zwingend vorgeschriebene Eltern- oder Kind-Rollen. Eine detaillierte Auflistung würde den Rahmen dieses Artikels sprengen, für eine aktuelle Übersicht sollten Sie die »WAI-ARIA 1.0 Authoring Practices« sowie die »ARIA-Techniken für WCAG 2.0« zu Rate ziehen.

Das Hinzufügen von ARIA-Attributen im Quelltext einer Seite ändert zunächst einmal nichts an deren Oberfläche. Wenn man einem

Für ein Beispiel der praktischen Anwendung von Landmark-Roles genügt ein Blick in den Quelltext dieser Seiten – wir setzen diese hier bei »Einfach für Alle« schon seit geraumer

sinnfreien DIV via ARIA die Rolle "checkbox" zuweist, so erkennt ein Screenreader zwar die Intention, das Verhalten einer Checkbox fehlt aber nach wie vor und muss per JavaScript nachgebildet werden. Auch in der grafischen Darstellung wird dieses Element weiterhin als leeres Element und nicht, wie man vielleicht meinen könnte, als Checkbox dargestellt. An diesem Beispiel sehen Sie schon, dass es oftmals besser wäre, die eventuell bereits vorhandenen passenden HTML-Elemente zu verwenden und nicht das Rad neu zu erfinden.

Das Elegante an dieser Lösung ist, dass erweiterte Bedienelemente in Widgets wie zum Beispiel Schieberegler den Nutzern genau so präsentiert werden, wie Schieberegler in Desktop-Anwendungen schon seit Jahren: es wird angesagt, dass es sich um einen Schieberegler handelt, der aktuell eingestellte Wert wird vorgelesen und die möglichen Werte, die der Nutzer (z.B. per Pfeiltaste) einstellen kann – der Lernaufwand zur Bedienung solcher Widgets in Webseiten geht für den Nutzer also gegen Null, dafür steigt die Zugänglichkeit um ein Vielfaches.

Die einzige negative Auswirkung auf Ihre Seiten ist, dass diese durch den Einsatz von WAI-ARIA nicht mehr validieren werden. Das ist aber in diesem Fall eher ein Problem des Validators, nicht Ihrer Nutzer oder der Browser, die diese verwenden.

Weitere Hintergründe dazu, was das W3C zur Entwicklung dieses Standards veranlasst hat und wie die weitere Planung verläuft finden Sie in der »Roadmap for Accessible Rich Internet Applications (WAI-ARIA Roadmap)«.

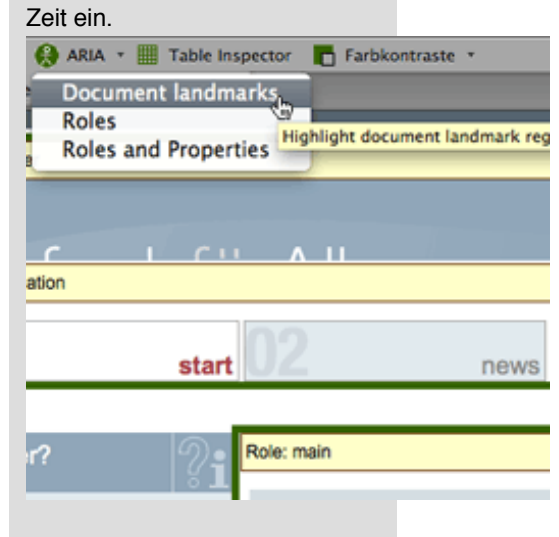
ARIA – Wer unterstützt das?

WAI-ARIA wird bereits seit einigen Versionsnummern von Screenreadern wie NVDA, VoiceOver, Window Eyes und JAWS sowie von Browsern wie Safari ab der Version 4, Internet Explorer ab Version 7 und Firefox ab v.1.5 unterstützt; weitere Hilfsmittel und Browser werden in Zukunft sicher folgen. Allerdings gibt es teils erhebliche Unterschiede in der Umsetzung, und selbst auf einer einzigen Plattform mit ein und demselben Screenreader kann es zu Unterschieden in der Nutzbarkeit kommen, je nachdem welchen Browser (z.B. Firefox oder IE) man benutzt. Hier kommen Sie also um einen praktischen Test mit echten Anwendern kaum herum.

Aktuelle Informationen über den Stand der Unterstützung in den diversen Hilfsmitteln finden Sie (mit viel Glück) auf den Seiten der jeweiligen Hersteller, z.B. »JAWS Support for ARIA« (Word-Datei). Sehenswert ist auch das Youtube-Video von Cannon Access, das zeigt, wie Screenreader-Nutzer mit ARIA-angereicherten Seiten interagieren: »ARIA Live Regions Screen Reader Demo«.

ARIA – Wie geht das?

Die einfachste Methode, um ARIA-Attribute in vorhandene HTML-Seiten hinein zu bekommen, ist per Skript. Wenn Sie bereits JavaScript-Bibliotheken wie z.B. jQuery einsetzen, dann ist dies mit einigen Zeilen Code erledigt – alles was Sie machen müssen ist, eindeutig identifizierbare IDs oder Klassen, die ausschließlich für bestimmte Inhalte verwendet werden, mit den entsprechenden Rollen-Attributen zu versehen:



Weitere Links zum Thema:

- cssgallery.info: »How screen readers speak a page with HTML5 and ARIA«
- [OpenAjax Accessibility](#): »OpenAjax Examples«
- [W3C](#): »WAI-ARIA Overview«
- zomigi.com: »Videos of screen readers using ARIA«

```
/* Skript zum Anhängen von WAI-ARIA-Rollen-Attributen mit jQuery: */
$(document).ready(function() {
  $("#kopf").attr("role", "banner");
  $("#nav").attr("role", "navigation");
  $("#skiplinks").attr("role", "navigation");
  $("#inhalt").attr("role", "main");
  $("#marginalie").attr("role", "complementary");
  $("#legende").attr("role", "note");
  $(".fehlerbox").attr("role", "alert");
});
```

Das Elegante an diesem Ansatz ist, dass der Validator nichts von den neuen, formal ungültigen Attributen mitbekommt und Ihre Seiten weiterhin validieren. Allerdings stellt sich bei uns ein gewisses Unbehagen ein, wenn man strukturelle Informationen wie ARIA-Landmarks an die Verhaltens-Schicht (und damit an das Vorhandensein von JavaScript) bindet – eigentlich gehören diese ins HTML.

Weiterhin problematisch an diesem Ansatz ist, dass diese Zuweisungen nicht zwangsläufig immer an den vorgesehenen Stellen greifen, weil nicht überall alleine anhand der ID entschieden werden kann, welche Rolle passt, oder ob das Element tatsächlich mit Inhalt befüllt ist. Im Zweifelsfall sollten Sie sich gegen die Verwendung solcher Skripte entscheiden und die benötigten ARIA-Attribute direkt ins HTML einfügen.

[role=complementary]

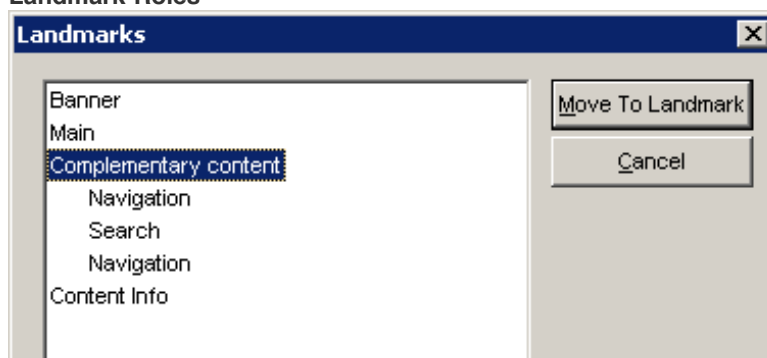
Eine Ausnahme von dieser Regel: wenn Widgets für ihre Funktionalität zwingend JavaScript voraussetzen, dann spricht selbstverständlich nichts dagegen, die ergänzenden ARIA-Informationen innerhalb dieser Widgets per JavaScript zu setzen. ARIA wurde ja gerade für den Einsatz in dynamischen Applikationen erdacht (wir erinnern uns: ARIA steht für »*Accessible Rich Internet Applications*«) und Werte müssen teilweise per JavaScript geändert werden (z.B. wenn ein Element auf- oder zugeklappt wird), um weiterhin sinnvoll zu sein.

```
<fieldset aria-expanded="true" class="klappe-auf">
  <legend>Dieses Fieldset wird angezeigt</legend>
  ...
</fieldset>

<fieldset aria-expanded="false" class="klappe-zu">
  <legend>Dieses Fieldset wird nicht angezeigt</legend>
  ...
</fieldset>
```

Hier nun eine Auswahl von verschiedenen ARIA-Attributen, die Sie auch heute schon ohne Nebenwirkungen in Ihren Seiten einsetzen können. Sie sollten auf jeden Fall vor dem unkritischen Einsatz abwägen, ob es nicht vielleicht doch bereits passendere HTML-Elemente und -Attribute gibt, die den Einsatz von ARIA überflüssig machen, weil sie vergleichbare semantische Information bereits in sich tragen:

Landmark-Roles



role="application"

Für Inhalte vom Typ Anwendung: »»A region declared as a web application, as opposed to a web document.«« . Diese Rolle sollte nur mit äußerster Vorsicht eingesetzt werden. Zum einen müssen dann sämtliche Inhalte (also auch statische Texte) mit Bedien-Elementen einer Anwendung verknüpft sein (z.B. über aria-labelledby oder aria-describedby), ansonsten werden sie von Screenreadern ignoriert. Zum anderen ändern Screenreader wie JAWS ihr Verhalten, wenn man dieses Attribut für das BODY-Element setzt: der Screenreader verhält sich dann wie bei einer Desktop-Anwendung und nicht wie bei Web-Inhalten.

role="banner"

Für den Kopfbereich einer Seite, der z.B. das Logo oder andere Objekte enthält, die sich auf die gesamte Site beziehen. Dieses Attribut sollte nur 1x je role="document" oder role="application" vergeben werden (diese können allerdings mehrfach pro Dokument vorkommen).

role="complementary"

Für ergänzende Zusatzinfos zum wesentlichen Inhalt der Seite.

role="contentinfo"

Für eine Kopf- bzw. Fußzeile, Sidebar o.ä. mit Metadaten, Impressum, Disclaimer, Hinweisen zum Datenschutz etc.

role="document"

Für Inhalte vom Typ Dokument: »»A region containing related information that is declared as document content, as opposed to a web application.«« In der Regel unnötig, weil der default-Zustand.

role="form"

Für Formulare: kann mehr enthalten als nur ein reines FORM-Element, sondern auch Hyperlinks oder Texte etc., die zum Formular gehören, aber ausserhalb von FORM stehen und hierüber zugeordnet werden.

role="main"

Für den wesentlichen Inhalt der Seite.

role="navigation"

Für jegliche eindeutig abgrenzbare Bereiche mit mehreren Navigations-Elementen.

role="search"

Für ein Such-Widget: Die Rolle sollte dem FORM-Tag zugewiesen werden, nicht den enthaltenen INPUT-Feldern.

Document-Structure-Roles**role="alert"**

Für Warnhinweise: »»A message with important, and usually time-sensitive, information.««

role="article"

Für Artikel im Sinne eines eigenständigen Teils der Seite oder der Anwendung, der für sich genommen eine abgeschlossene Sinneinheit bildet. Vorsicht: "article" ist nicht im Sinne eines redaktionellen Artikels zu verstehen (das wäre "story", diese Rolle gibt es jedoch nicht), sondern eher vergleichbar mit einem Artikel in einem Regal.

role="columnheader"

Für Spaltenköpfe bei Inhalten, die tabellarische Eigenschaften haben.

role="definition"

Für Definitionen: »»A definition of a term or concept.«« Unnötig, wenn DFN oder DL verwendet wird.

role="directory"

Für ein Verzeichnis im Sinne eines Inhalts- oder Dateiverzeichnisses: »»A list of references to members of a group, such as a static table of contents.««

role="group"

Für generische Gruppen: »»A set of user interface objects which are not intended to be included in a page summary or table of contents by assistive technologies. Contrast with region which is a grouping of user interface objects that will be included in a page summary or table of contents.««

role="img"

Für Abbildungen: »»An img can contain captions and descriptive text, as well as multiple image files that when viewed together give the impression of a single image.««

role="list"

Für Elemente, deren enthaltene Elemente den Charakter einer Auflistung haben: »»A group of non-interactive list items. Also see listbox. Lists contain children whose role is listitem, or elements whose role is group which in turn contains children whose role is listitem.««

role="listitem"

Für Listenelement einer solchen Liste: »»A single item in a list or directory. Authors MUST ensure elements with role listitem are contained in, or owned by, an element with the role list.««

role="math"

Für mathematische Formeln: »»Content that represents a mathematical expression.««

role="note"

Für Einschübe und Ergänzungen: »»A section whose content is parenthetical or ancillary to the main content of the resource.««

role="presentation"

Für rein dekorative Elemente, die z.B. in der Sprachausgabe keine semantische Bedeutung haben sollen: »»An element whose implicit native role semantics will not be mapped to the accessibility API. The intended use is when an element is used to change the look of the page but does not have all the functional, interactive, or structural relevance implied by the element type, or may be used to provide for an accessible fallback in older browsers that do not support WAI-ARIA.«« Dies heisst jedoch nicht, dass die Inhalte eines Elements mit der Rolle "presentation" im Screenreader nicht vorgelesen werden – im Gegenteil. Lediglich das derart ausgezeichnete Element (z.B. eine Überschrift) taucht nicht im Dokumentenbaum auf, enthaltene Texte jedoch weiterhin.

role="radiogroup"

Für eine Gruppe von Radiobuttons: wird benötigt um z.B. am umschließenden Element `aria-required="true"` setzen zu können und muss `role="radio"` enthalten.

role="region"

Für eine Region einer Seite als Fallback, falls andere Landmark Roles nicht passen. »»A large perceivable section of a web page or document, that the author feels is important enough to be included in a page summary or table of contents, [...] Authors SHOULD ensure that a region has a heading referenced by `aria-labelledby`. [...] When defining regions of a web page, authors are advised to consider using standard document landmark roles. If the definitions of these regions are inadequate, authors can use the region role and provide the appropriate accessible name.««

role="row"

Für Tabellenzeilen bei Inhalten, die tabellarische Eigenschaften haben.: »»A row of cells in a grid.««

role="rowheader"

Für Zeilenköpfe bei Inhalten, die tabellarische Eigenschaften haben.: »»A cell containing header information for a row in a grid.««

role="separator"

Für Trennelemente, z.B. verschiebbare Stege zwischen Bereichen eines Widgets oder einer Anwendung: »»This is a visual separator between sections of content. For example, separators are found between groups of menu items in a menu or as the moveable separator between two regions in a split pane.««

role="toolbar"

Für Werkzeugleisten (z.B. die Button-Leiste eines Texteditors): »»A collection of commonly used function buttons represented in compact visual form.««

role="tree"

Für Baum-Strukturen: »»A type of list that may contain sub-level nested groups that can be collapsed and expanded.««

role="treeitem"

Für Baumelemente: »»An option item of a tree. This is an element within a tree that may be expanded or collapsed if it contains a sub-level group of treeitems.««

Zustände & Eigenschaften (States & Properties)**aria-autocomplete="list"**

Element hat eine ihm zugeordnete Liste von Vorschlägen, mit denen der eingegebene Text komplettiert werden kann. »»A list of choices appears from which the user can choose, but the edit box retains focus.««

aria-describedby="#ID"

Verknüpft ein Element mit anderen (vorzugsweise textlichen) Elementen, die das Objekt beschreiben. Beispiel: ``

aria-expanded="false" / aria-expanded="true"

Universal-Attribut zur Kennzeichnung auf- bzw. zugeklappter Elemente: »»Indicates whether the element, or another grouping element it controls, is currently expanded or collapsed.««

aria-haspopup="true"

Zur Kennzeichnung, dass dem Element ein Pop-Up zugeordnet ist: »»Indicates that the element has a popup context menu or sub-level menu.««

aria-labelledby="ID_der_Beschreibung"

Für die Zuordnung einer Beschriftung zu einem Element: »»Identifies the element (or elements) that labels the current element««) Beispiel: `<div id="datepicker" role="grid" aria-labelledby="abflugdatum">`

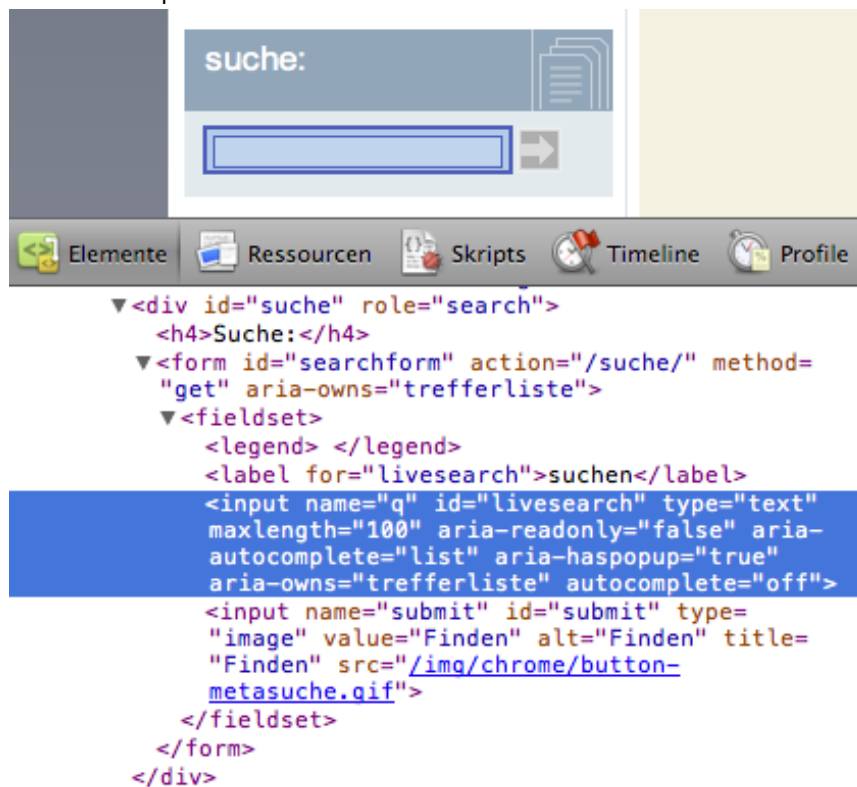
aria-live="assertive" / aria-live="polite"

Ergänzende Rolle zu `role="alert"` (eine Live-Region, die Aktualisierungen innerhalb einer Seite anzeigt: »»Indicates that an element will be updated, and describes the types of updates the user agents, assistive technologies, and user can expect from the live region.««

aria-owns="ID"

Zur Zuordnung von weiteren Elementen auf einer Seite, die ansonsten nicht über das DOM hergestellt werden kann: »»Identifies an element (or elements) in order to define a visual, functional, or contextual parent/child relationship between DOM elements where the DOM hierarchy cannot be used to represent

the relationship.««



aria-required="true"

Kennzeichnet Pflichtfelder: »»Indicates that user input is required on the element before a form may be submitted.«« Beispiel: `<textarea id="foo" name="bar" class="baz" cols="15" rows="5" aria-required="true">`

CSS zum Debugging

Zum Abschluss noch ein kleiner CSS-Kniff, um die Struktur einer mit WAI-ARIA angereicherten Seite zu visualisieren und eventuelle Fehler zu finden. Zunächst wird allen Elementen (*) mit dem Attribut role ([role]) eine kleine Box nach Art eines Tooltips vorangestellt (:before). In dieses Tooltip wird der Inhalt des Rollen-Attributs per Generated Content geschrieben (content: "[role=" attr(role) "];), der Rest sind optische Formatierungen:

```
/* Rollen anzeigen ----- */

*[role]:before {
  content: "[role=" attr(role) "];";
  color: #333;
  background: #fdfddc;
  outline: 1px solid #999;
  padding: .2em;
  -webkit-box-shadow: 2px 2px 2px #ccc;
  -moz-box-shadow: 2px 2px 2px #ccc;
  box-shadow: 2px 2px 2px #ccc;
}
```

Dann werden verschiedene Rollen mit einer farbigen Outline versehen, die den gesamten Bereich umfasst:

```
/* unterschiedliche Rahmenfarben je nach Rollen-Typ ----- */

*[role] {
  outline: 1px solid gray;
}

*[role=banner] {
  outline: 1px solid red;
}

*[role=navigation] {
  outline: 1px solid green;
}

*[role=main] {
  outline: 1px solid blue;
}
```

Als letztes werden die unterschiedlichen Eigenschaften per Outline hervorgehoben. Leider gibt es in CSS keinen Wildcard-Character (also etwas wie `*[aria-*)`), um sämtliche Attribute abzufangen, die mit `aria-` beginnen, daher müssen diese leider einzeln aufgelistet werden:

```
/* States & Properties anzeigen ----- */

/* geht leider nicht:
*[aria-*) {
  outline: 1px dashed red;
} */

*[aria-autocomplete],
*[aria-describedby],
*[aria-expanded],
*[aria-haspopup],
*[aria-labelledby],
*[aria-live],
*[aria-owns],
*[aria-readonly],
*[aria-required] {
  outline: 1px dashed red;
}
```

Sollte man HTML5 & ARIA zusammen verwenden?

Es gibt eine ganze Reihe guter Beispiele, in denen die erweiterte Semantik von HTML5 und die hilfsmittel-spezifischen Eigenschaften von WAI-ARIA erfolgreich kombiniert werden, inklusive Testcases zur Überprüfung der Machbarkeit und den passenden Workarounds. Allerdings stellt sich für uns schon die Frage nach der Sinnhaftigkeit dieser Kombination: HTML5 wird entwickelt, um die Lücken endgültig zu schließen, welche die für den heutigen Stand der Technik unzureichende HTML4-Spezifikation hinterlassen hat; WAI-ARIA hingegen ist als Übergangslösung gedacht, um Hilfsmitteln den Zugriff auf interaktive Elemente zu ermöglichen, die ohne zusätzliche Informationen unzugänglich blieben. Wenn nun HTML5 zusammen mit ARIA eingesetzt wird, dann kann dies im Umkehrschluss eigentlich nur bedeuten, dass es auch in HTML5 noch Lücken gibt, wo z.B. Screenreader nicht mit den für sie nötigen Informationen versorgt werden – ein Zustand, der eigentlich durch HTML5 behoben sein sollte.

Eine durchaus nachvollziehbare Begründung für den kombinierten Einsatz ist die Unterstützung älterer Browser und/oder älterer Hilfsmittel, die eventuell noch nicht den vollen Satz der Möglichkeiten verstehen und korrekt umsetzen. Derek Featherstone beschreibt dies im Artikel »ARIA and Progressive Enhancement« bei A List Apart anhand des NAV-Elements aus HTML5 in Kombination mit der Rolle "navigation" aus WAI-ARIA:


```
<nav role="navigation">
  <ul>
    <li><a href="/">Home</a></li>
    <li><a href="/archives/">Archives</a></li>
    <li><a href="/about/">About</a></li>
    <li><a href="/contact/">Contact</a></li>
  </ul>
</nav>
```

Was zunächst aussieht wie ein unnötiger Doppler ergibt bei näherer Betrachtung durchaus Sinn: weder HTML5 noch WAI-ARIA werden, global betrachtet, von sämtlichen Browsern und allen Hilfsmitteln vollständig und korrekt unterstützt. Browser/Screenreader-Kombinationen, in denen NAV bereits verstanden wird, können diese Informationen zur Verfügung stellen; andere Kombinationen, in denen zumindest ARIA unterstützt wird, erhalten diese Informationen über die ARIA-Rolle. Wenn keines von beiden verstanden wird (NAV also wie ein generisches DIV ohne weitere Attribute behandelt wird), dann bekommt der Nutzer trotzdem die gleichen Informationen wie bei klassischen HTML4/XHTML1-Seiten über die Auszeichnung der Navigationspunkte als Liste.

Problematisch wird dieser Ansatz nur, wenn durch solche Doppler Informationen tatsächlich mehrfach vorgelesen werden. Leider kann man weder server- noch client-seitig abfragen, ob und wenn ja welcher Screenreader an der Accessibility-Schnittstelle hängt, geschweige denn welchen Grad der Unterstützung dieser für diverse technische Möglichkeiten hat. Also scheitern an dieser Stelle auch alle Ansätze des »*Progressive Enhancement*«, da man nicht testen kann, ob ein User Agent ein Objekt in der gewünschten Form versteht, um dann im Code darauf zu reagieren – Tests hierzu und mögliche Workarounds finden Sie im oben genannten »A List Apart«-Artikel.

Hinzu kommt, dass Screenreader vergangener und auch aktueller Baureihen teilweise erhebliche Bugs in der Umsetzung von HTML5 & WAI-ARIA haben. Ob das nun am Screenreader, an der Accessibility-Schnittstelle des Betriebssystems oder am verwendeten Browser liegt, ist aus Sicht eines Webentwicklers eigentlich egal: ändern kann man alle drei nicht.

Nun könnte man zum Umschiffen von Bugs in Screenreadern zusätzliche Elemente einfügen, wenn z.B. Screenreader wie Window Eyes mit dem obigen Code nicht zurechtkommen. Der folgende Code hilft auch diesem auf die Sprünge:

```
<div role="navigation">
  <nav>
    <ul>
      <li><a href="/">Home</a></li>
      [...]
    </ul>
  </nav>
</div>
```

Allerdings haben wir hier nun das Problem, dass der Navigationsbereich in anderen Hilfsmittel-/Browser-Kombinationen zweifach vorgelesen wird (im konkreten Fall passiert dies in der Tat bei NVDA 2010.2 & Firefox 4); User Agents, die sich ihrerseits an die Standards halten, werden dadurch also effektiv abgestraft – *das kann auch nicht das Ziel sein*.

Hier kann man also nur auf die Nutzer fehlerhafter Hilfsmittel hoffen, dass diese Upgrades möglichst zeitnah vollziehen, um die erweiterten Möglichkeiten nutzen zu können. Trotzdem: nicht jeder Screenreader-Nutzer hat die Möglichkeit, auf moderne Programme wie NVDA und Firefox umzusteigen, die moderne Webstandards hervorragend unterstützen. Gerade im beruflichen Umfeld ist die Kombination aus JAWS 6 und Internet Explorer 6 noch gelegentlich anzutreffen, allerdings kann man nicht erwarten, damit moderne Webapplikationen auch nur im Ansatz nutzen zu können. Anbieter können nur das größtmögliche an Zugänglichkeit einbauen; wenn dies dann nicht unterstützt wird und auch Fallback-Lösungen scheitern (oder prinzipienbedingt nicht möglich sind), dann gibt es nicht mehr viel, was man als Anbieter machen kann – der Ball liegt also hier bei den Anwendern. Bruce Lawson, Accessibility Evangelist bei Opera, formuliert dies wie folgt:

»... if you correctly use a W3C specification, like this one, and obey WCAG, then I believe that your responsibility as a web author is done. I won't delay using a technique because a user agent can't deal with it (although I will try to give extra help where I can).«

<zitat>

Alles im Fluss – Formularlayout mit CSS

Viele große Websites, die ohne Layout-Tabellen gestaltet wurden, bedienen sich der float-Eigenschaften des CSS-Box-Modells. Bekannte CSS-Frameworks wie YAML bauen ebenfalls auf diesen Methoden auf und erreichen damit höchst flexible Layouts. Das Box-Modell beschreibt die Eigenschaften von CSS 2, beliebige Inhalts-»Behälter« (Boxen) einer HTML-Seite zu formatieren. Hiermit können Sie Inhaltsblöcke, die im rohen HTML nacheinander stehen würden, in der grafischen Darstellung nebeneinander positionieren. Wenn Sie nun eine solche Box per `float: positionieren`, so wird diese aus dem »Fluss« des Dokumentes herausgenommen und kann nun durch die Angabe `float: right;` oder `float: left;` beliebig rechts oder links neben einer darauf folgenden Box positioniert werden. Die Nachbar-Boxen umfließen nun diese Box. So lassen sich mit nur wenigen Zeilen Code komplexe mehrspaltige Layouts umsetzen, wie Sie hier an den »Einfach für Alle«-Seiten sehen können.

Genau diese Eigenschaften können Sie auch ausnutzen, um in Formularen die Labels vollkommen tabellenfrei neben den dazu gehörigen Kontrollelementen zu positionieren. Die einzige Schwierigkeit besteht darin, dass die Labels einiger Kontrollelemente wie Textfelder oder Auswahlmenüs üblicherweise vor dem Element (also links daneben oder darüber) stehen; bei anderen Kontrollelementen wie Radiobuttons und Checkboxes stehen diese nach dem Element (d.h. rechts daneben). Sie sollten sich in Ihren Formularen unbedingt an diese Konvention halten, weil dies den Erwartungen des Nutzers im allen Betriebssystemen entspricht.

Eine detailliertere, dafür aber auch sehr technische Beschreibung der box- und float-Modelle finden Sie in den CSS 2-Spezifikationen des W3C; wem die Lektüre der W3C-Spezifikation zu trocken ist: die wohl beste Einführung in dieses Thema ist der Artikel »Float: the theory« von Holly Bergevin und John Gallant, hier in der deutschen Übersetzung »Float: Die Theorie«

Ein Beispielformular

Erst mal das HTML

Das folgende Formular enthält die gängigen Elemente zur Eingabe von Text und zur Auswahl von Optionen. Als Vorbereitung auf die Formatierung per CSS wurden bereits einige Klassenangaben eingefügt. Alle Elemente, bei denen das Label links vom Kontrollelement steht, haben die Klasse `"left"`, alle Elemente, bei denen das Label rechts daneben steht, haben die Klasse `"right"`:

```

<form action="foo">
  <fieldset>
    <legend>Beispielformular</legend>

    <label for="textfield" class="left">Text:</label>
    <input type="text" id="textfield" name="textfield" size="12"><br>

    <label for="auswahl" class="left">Auswahl:</label>
    <select id="auswahl">
      <option value="auswahl1">Auswahl 1</option>
      <option value="auswahl2">Auswahl 2</option>
    </select><br>

    <input type="radio" id="radio1" name="radio" value="entweder" class="right">
    <label for="radio1">Entweder</label><br>

    <input type="radio" id="radio2" name="radio" value="oder" class="right">
    <label for="radio2">Oder</label><br>

    <input type="checkbox" id="check1" value="sowohl" class="right">
    <label for="check1">Sowohl</label><br>

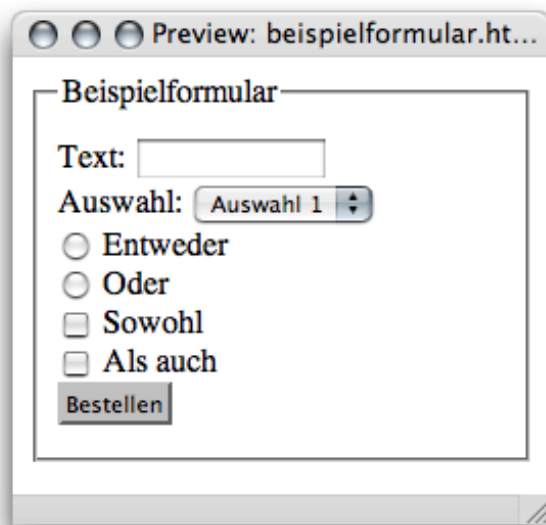
    <input type="checkbox" id="check2" value="alsauch" class="right">
    <label for="check2">Als auch</label><br>

    <button type="submit" class="right">Bestellen</button>

  </fieldset>
</form>

```

Das gerenderte Ergebnis zeigt ein Formular ohne jegliche Formatierung, aber in einer für nicht-grafische Darstellungsformen optimal verarbeitbaren Form:



Dann das CSS

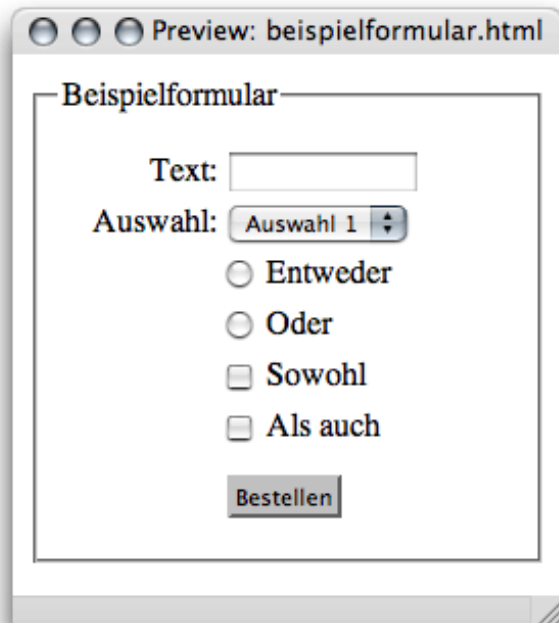
Im nächsten Schritt wird das Style Sheet erstellt, um die Labels und Kontrollelemente zu positionieren. Zunächst werden die Labels mit der Klasse "left" mit einer Breite versehen und rechtsbündig an eine imaginäre Mittelachse gebracht. Über `margin-right` wird der Abstand der Labels zu den daneben stehenden Kontrollelementen definiert:

```
.left {  
  float: left;  
  text-align: right;  
  width: 40%;  
  margin-right: 2%;  
}
```

Dann kommen die Radiobuttons und Checkboxes an die Reihe, die im HTML bereits mit der Klasse "right" versehen wurden. Der Abstand ergibt sich hier aus der Breite der Textlabels für die vorhergehenden Textfelder (40%) plus des Abstands dieser Labels zu ihren Kontrollelementen (2%). Hieraus ergibt sich für die folgenden Kontrollelemente inklusive des »Absenden«-Buttons ein linker Abstand von $40\% + 2\% = 42\%$, um sich an besagter Mittelachse auszurichten:

```
.right, button {  
  margin-left: 42%;  
}
```

Das war's auch schon – mit ein paar Zeilen Code (Beispieldatei: bespielformular.html) haben sie nun eine Basis-Formatierung für sämtliche Formulare Ihrer Website. Das Ergebnis sehen Sie in den folgenden Screenshots:



Preview: beispielformular.html

Beispielformular

Text:

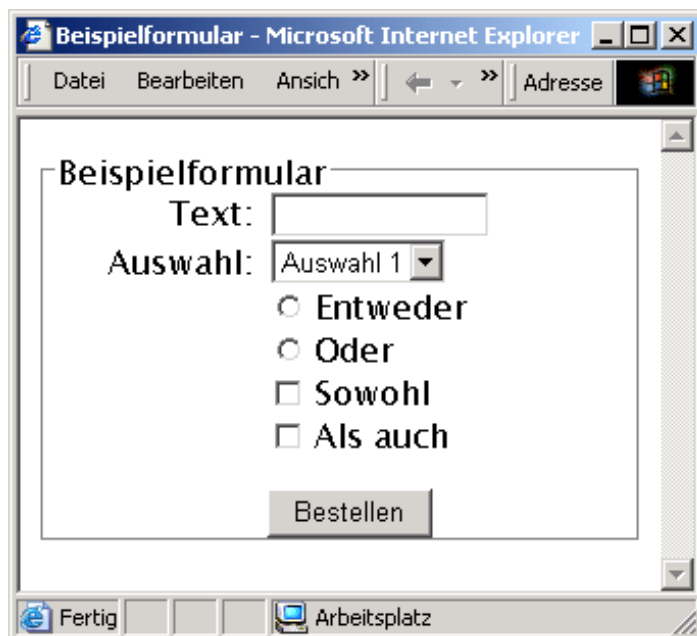
Auswahl:

☐ Entweder

☐ Oder

☐ Sowohl

☐ Als auch



Beispielformular - Microsoft Internet Explorer

Datei Bearbeiten Ansicht >> << >> Adresse

Beispielformular

Text:

Auswahl:

☐ Entweder

☐ Oder

☐ Sowohl

☐ Als auch

Fertig Arbeitsplatz

Luxuriöse Buttons

In den vorherigen Beispielen haben wir zum Absenden des Formulars `<button>`-Tags benutzt. In der unformatierten Variante sind diese eher hässlich und erinnern auch in modernen Browsern stark an die Zeiten von Windows 3.11. Vielleicht verwenden Frontend-Entwickler deshalb oft Buttons vom Typ `<input type="submit">`, ärgern sich dann über dessen mangelhafte Anpassbarkeit und greifen am Ende doch wieder auf `<input type="image">` zurück.

Wesentlich flexibler ist man mit dem `BUTTON`-Element, da dieses nicht nur sehr individuell gestaltbar ist, sondern auch noch so gut wie alle anderen HTML-Elemente enthalten darf. Die Buttons selbst können zum Beispiel in einer Auflistung stehen, und selbst auch Block-Level-Elemente wie `H1-H6` oder `P` und selbstverständlich auch Inline-Elemente oder Grafiken mit Alternativtext enthalten:

```
<fieldset>
  <legend>Bitte wählen Sie:</legend>
  <ol>
    <li>
      <button type="submit" name="karte" value="sued">
        <h3>Südkurve</h3>
        <p><strong>10 &euro;</strong></p>
        <p>(Stehplatz)</p>
      </button>
    </li>
    <li>
      <button type="submit" name="karte" value="nord">
        <h3>Nordkurve</h3>
        <p><strong>20 &euro;</strong></p>
        <p>(Sitzplatz)</p>
      </button>
    </li>
    <li>
      <button type="submit" name="karte" value="ost">
        <h3>Osttribüne</h3>
        <p><strong>30 &euro;</strong></p>
        <p>(Sitzplatz)</p>
      </button>
    </li>
    <li>
      <button type="submit" name="karte" value="west">
        <h3>Westtribüne</h3>
        <p><strong>50 &euro;</strong></p>
        <p>(VIP-Loge)</p>
      </button>
    </li>
  </ol>
</fieldset>
```

Bitte wählen Sie:

- Südkurve**
10 €
(Stehplatz)
- Nordkurve**
20 €
(Sitzplatz)
- Osttribüne**
30 €
(Sitzplatz)
- Westtribüne**
50 €
(VIP-Loge)

Mit ein wenig CSS können Sie aus diesen gänzlich unattraktiven Buttons nun das folgende zaubern:

Bitte wählen Sie:



Das elegante an dieser Lösung ist, dass sie gänzlich ohne Hintergrundbilder auskommt. Die Unter-Elemente der Buttons können wie im Beispiel einzeln per CSS mit Verläufen, Schlagschatten, runden Ecken etc. versehen werden, ohne die Geschwindigkeit der Seiten durch zusätzliche Requests für Grafikdateien zu verringern. Den kompletten Code finden Sie in der Bespieldatei `button.html`.

Wenn Sie schon dabei sind Ihre Buttons direkt im CSS statt in Photoshop zu stylen, sollten Sie im Sinne der Zugänglichkeit auch an die möglichen Zustände von Buttons `hover`, `focus` & `active` denken und für diese Zustände das Aussehen festlegen. Gerade der aktive Zustand wird gerne vergessen, dabei ist dieser recht einfach zu erreichen, indem man z.B. den enthaltenen Verlauf einfach um 180° dreht, oder indem man den Button in gedrücktem Zustand um 1 Pixel nach unten oder nach unten rechts versetzt:

Eine gute Anleitung zum Erstellen von reinen HTML- & CSS-Buttons finden Sie im Artikel »CSS3 Gradient Buttons«; auch für den umgekehrten Fall, nämlich wenn Buttons in der Optik von Links präsentiert werden sollen (z.B. um eine sekundäre Aktion weniger prominent zu zeigen), gibt es ein passendes Tutorial: »Styling buttons to look like links«

```
button:active {
  position: relative;
  top: 1px;
  left: 1px;
}
```

Browserbugs und andere Unterschiede im Rendering

Styling von Kontrollelementen

Im Gegensatz zu Buttons und Texteingabefeldern, die sich hervorragend stylen lassen, sollten Sie bei anderen Elementen eines Formulars lieber die Finger davon lassen, weil das Styling z.B. von Radiobuttons und Checkboxes in der Regel schief geht. Auch aus Sicht der Barrierefreiheit stellt sich für uns die Frage, ob es sinnvoll ist, Webseiten und ihre Elemente in allen Browsern gleich aussehen zu lassen.

Wenn die optische Darstellung von Formularelementen zu sehr von der für den Nutzer gewohnten Darstellung in seinem jeweiligen Betriebssystem abweicht, dann leidet darunter naturgemäß auch die Wiedererkennbarkeit und die Erlernbarkeit.

Falls Sie es doch wagen wollen, hier ein paar Links zu Artikeln, in denen die negativen Folgen illustriert werden:

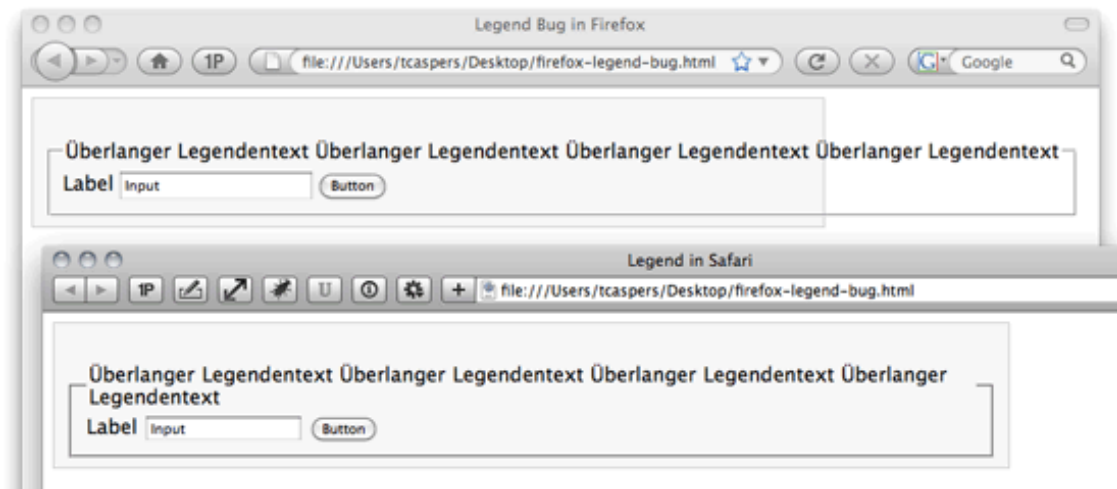
- »Verrückte Formulare«
- »Styling form controls with CSS, revisited«
- »Form elements – checkbox, radio button and CSS«

Legendäre Browser-Bugs

Auch wenn wir zu Anfang die Verwendung von `FIELDSET` und `LEGEND` empfohlen haben – der Ehrlichkeit halber muss gesagt werden, dass insbesondere die Verwendung des `LEGEND`-Elements nur etwas für ganz hartgesottene Frontend-Entwickler ist. Leider gibt es hier eine ganze Reihe Inkonsistenzen und Bugs in verschiedenen Browsern, die einen sinnvollen Einsatz manchmal fast unmöglich machen. Besonders

ärgerlich hierbei ist, dass diese Bugs teilweise seit mehreren Jahren bekannt und dokumentiert sind, es aber anscheinend niemand für nötig befindet, diese zu beheben.

So ist es bis heute kaum möglich, dass lange Legendentexte umbrechen und bei Bedarf mehrzeilig dargestellt werden, ohne dass es in dem einen oder anderen Browser zu Problemen kommt (Beispieldatei: `firefox-legend-bug.html`). Firefox erlaubt erst ab Version 3.6 die Deklaration einer Breite für `<legend>...</legend>`, bis einschließlich Firefox 3.5.7 ist die Breite der Legende immer gleich der Breite des enthaltenen Textes.



Eine Empfehlung, möglichst kurze Legendentexte zu verwenden ist sicher sinnvoll, aber spätestens bei Formularen in schmalen Spalten braucht man eine robustere Lösung. Diese wird jedoch dadurch erschwert, dass `LEGEND` in eigentlich allen Browsern nicht wie ein normales Element auf CSS-Anweisungen reagiert. Oft hilft nur, ein zusätzliches Element (z.B. `SPAN`) in `<legend>` zu schachteln und dann die Formatierung über das innere Element vorzunehmen.

```
<fieldset>
  <legend><span>Legende</span></legend>
  ...
</fieldset>
```

Im CSS wird dem inneren `` nun ein `display:block;` zugewiesen, wodurch sich die Hintergrundfarben immer auf die volle Breite des Fieldsets erstrecken.

Weitere Tipps & Tricks zum Thema Legendentexte:

- »Line wrapping text in legend elements«
- »Display: block and form legend elements«

Gestalterische Hilfen für Nutzer mit Behinderung

Deutlicher Fokus-Indikator bei Maus- und Tastaturnavigation

In Formularen wird generell, also auch von Nutzern ohne motorische Behinderung oder Sehbehinderung, öfter per Tastatur navigiert, weil der Nutzer in der Regel schon beide Hände auf der Tastatur hat und der Griff zur Maus eine Unterbrechung darstellen würde. Daher ist hier die geräteunabhängige Wahrnehmbarkeit und Bedienbarkeit noch wichtiger als bei reinen Textdokumenten ohne Formulare. Problematisch ist dies besonders bei mehrspaltigen Formularen: hier sollten Sie unbedingt auf eine Abfolge der Felder und Fieldsets im Quelltext achten, die dem optischen Ablauf des Formulars entspricht. Gleichzeitig muss dem Tastaturnutzer ein sehr deutlich erkennbarer optischer Indikator gegeben werden, wo sich der aktuelle Fokus befindet, da man die aktuelle und die nächste Position nicht zwangsläufig vom Aufbau des Formulars herleiten kann.

Mit modernem CSS lässt sich die Gestaltung der Formularelemente auch ohne den Einsatz von JavaScript verändern. So können Sie zum Beispiel die Hintergrund- oder Rahmenfarbe von Textfeldern ändern, sobald diese aktiviert sind. Damit teilen Sie dem Benutzer deutlich mit, in welchem Abschnitt des Formulars er sich

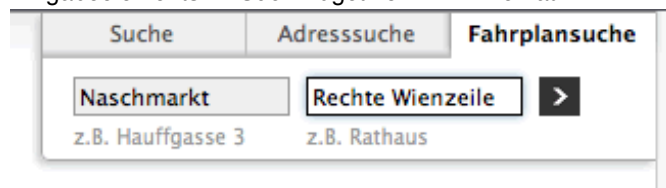
zurzeit befindet und welche Eingabe als nächstes erwartet wird. Hierzu müssen Sie zunächst einmal die Hintergrund- und ggf. Rahmenfarbe der Textfelder definieren:

```
input[type="text"], textarea {
  background: #ddd;
  border: 1px solid #ccc;
}
```

Als nächstes werden die Textfelder beim Erreichen des Fokus (d.h. durch das Setzen der Einfügemarke per Tabulator oder Maus oder durch Anklicken des verknüpften Labels) farblich hervorgehoben:

```
input[type="text"]:focus, textarea:focus {
  color: #000;
  background: #fff;
  border: 1px solid #000;
  outline: 1px solid blue;
}
```

Ein Beispiel für diese Technik ist der per CSS-Outline erreichte Halo-Effekt beim Fokussieren eines Eingabeelements im Suchwidget von www.wien.at:



Wie Sie einen vergleichbaren Effekt erreichen können, beschreibt der Artikel »Mit CSS3 aktive Formularfelder auf Websites deutlicher kennzeichnen«.

Zeig mir deine Label

Gerade bei umfangreichen Formularen mit einer Vielzahl von Kontrollelementen kann es sinnvoll sein, das aktive Label bei der Interaktion durch den Nutzer hervorzuheben, wie im nebenstehenden Screenshot zu sehen ist:

1 How do you want to use

☐ Private

☒ **Business**

☐ Education

☐ Don't know

☐ Other:

Auch dies ist mit einigen Zeilen Code zu bewerkstelligen:

```
label:hover, label:focus {
  background-color: #ccc;
  outline: 1px solid #999;
}
```

Bitte nicht nachmachen!

Sie sollten auf keinen Fall den Fokus-Indikator wegnehmen. Diese Maßgabe betrifft sowohl das in bestimmten Redaktionssystemen verbreitete `onfocus="this.blur()"`, das per Tastatur fokussierte Elemente sofort per Skript wieder de-fokussiert, als auch die sogenannten Reset-Styles, die oft unkritisch eingesetzt werden. Sie führen dazu, dass Wahrnehmbarkeit und Orientierung auf den Seiten massiv leiden.

Zwar steht in einem der verbreitetsten Reset-Bausteine extra noch ein Kommentar, welcher Anwender darauf hinweist, dass sie die Regeln für `:focus` noch ergänzen müssen. Wie die praktische Erfahrung gezeigt hat, wird dies dann aber gerne mal vergessen. Und so findet, wie Dirk Jesse in seinem Artikel »Löschen ist keine

Lösung« zutreffend bemerkt, das folgende »Reset CSS«-Codeschnipsel seinen Weg in verbreitete CSS-Frameworks:

```
/* remember to define focus styles! */
:focus {outline: 0; }
```

Der Erfinder dieses Reset-Bausteins, Eric Meyer, hatte mittlerweile ein Einsehen und hat den Code so geändert, dass die Fokus-Indikatoren nicht mehr automatisch unterdrückt werden. Mehr dazu bei »Reset Revisited« – bleibt zu hoffen, dass der geänderte Code ebenso schnell Verbreitung finden wird wie der Ursprüngliche.

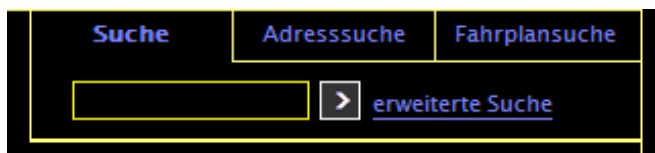
Optische Unterstützung bei geänderten Farbeinstellungen

Viele sehbehinderte Nutzer sind mit individuellen Einstellungen ihres Rechners und des Browsers im Netz unterwegs. Dazu gehören stark vergrößerte, aber auch verkleinerte Schriften und individuelle Farbeinstellungen, um die bei einigen Formen der Sehbehinderung auftretenden Blendeffekte zu mildern.

Hierfür bieten sämtliche Betriebssysteme eigene Einstellungsmöglichkeiten. Bei Linux-Desktops und unter Windows werden eine ganze Reihe so genannter Kontrastschemata mitgeliefert, die radikal den gesamten Bildschirminhalt verändern. Verbreitet ist die invertierte Darstellung z.B. mit gelbem Text auf schwarzem Hintergrund (unter Windows zu finden in den Systemeinstellungen als »Kontrast #1«). Unter Mac OS X hingegen kann entweder der gesamte Bildschirm in Graustufen oder invertiert (vergleichbar einem Foto-Negativ) dargestellt werden; der Bildschirm-Kontrast kann zudem systemweit mit einem Schieberegler sehr genau an die jeweiligen Bedürfnisse angepasst werden.

Problematisch werden solche Einstellungen für den Nutzer, wenn wie im Falle der Windows-Kontrastschemata sämtliche optischen Hilfen verschwinden, die der Designer eingebaut hat. Das System bzw. der Browser ignoriert in diesem Fall sämtliche Hintergrundfarben und stellt stattdessen überall die vom Nutzer eingestellte Hintergrundfarbe dar. Im konkreten Anwendungsfall wären also die Hilfen in Fieldsets oder die in unseren Designbeispielen zu sehenden gestreiften Hintergründe unsichtbar.

Aber es gibt einen Ausweg: Windows entfernt zwar alle Hintergrund-Farben (in CSS: `background-color`), nicht jedoch die Ränder (wie z.B. `border: 2px solid #000;`). Diese Tatsache kann man nutzen und seinen Farbflächen einen kleinen Rand in der gleichen Farbe geben – diese Ränder können dann für sehbehinderte Nutzer eine wertvolle Orientierungshilfe geben und fallen dem normalsichtigen Nutzer nicht weiter auf.



Weitere Informationen zu diesem Thema finden Sie im Artikel »Image Replacement-Techniken nicht zugänglich für Sehbehinderte« und in der BITV-Reloaded-Serie bei Anforderung 2 »Verständlichkeit ohne Farbe«.

Klick mich, ich bin ein Label

Mit ein wenig CSS können und sollten Sie Ihre Besucher informieren, dass man die Labels auch tatsächlich anklicken kann. Dies kann durch eine grafische Hervorhebung und durch eine Veränderung des Cursors beim Überfahren mit der Maus geschehen. Durch das folgende Beispiel verwandelt sich der Mauszeiger beim Überfahren des Labels in eine Hand mit Zeigefinger:

Auch die WCAG 2.0 hat zu diesem Punkt eine deutliche Meinung – wie man diesen Punkt erfüllt lesen Sie in der deutschen Übersetzung der Techniken unter:

- G183: Benutzung eines Kontrastverhältnisses von 3:1 mit umgebendem Text und Bereitstellung zusätzlicher visueller Hinweise für Links oder Steuerelemente bei darauf liegendem Fokus, wenn nur Farbe benutzt wird, um diese zu identifizieren
- G195: Benutzung einen hoch sichtbaren Fokus-Indikator, der vom Autor bereitgestellt wird

```
label, select,  
input[type=checkbox],  
input[type=radio],  
input[type=button],  
input[type=submit] {  
    cursor: pointer;  
}
```

Falls Sie ein IE-spezifisches Style Sheet benutzen, sollten Sie an dieser Stelle noch die Angabe `cursor:hand`; deklarieren, da ältere Versionen des Internet Explorer die W3C-konforme Variante nicht verstehen.

Eingabefelder vom Typ Text (`<input type="text">` bzw. `<textarea>`) benötigen hingegen keine Sonderbehandlung im CSS, obwohl dies mit dem Selektor `input[type=text]` möglich wäre – sämtliche uns bekannten Browser stellen hier von ganz allein eine Einfügemarke als Cursor dar.

Die Labels und die Kontroll- bzw. Eingabefelder werden in den gängigen Browsern durch eine Outline hervorgehoben, sobald sie mit der Tabulatortaste erreicht werden. Begehen Sie jedoch nicht den Fehler und nehmen diesen Fokus per JavaScript wieder weg, wie es viele Standard-Installationen des Open-Source-CMS Typo3 tun. Solche Formulare sind leider per Tastatur oder mit anderen alternativen Zeige- und Eingabegeräten nicht mehr zu bearbeiten.

Kennzeichnung inaktiver Felder (Disabled & Readonly)

Ein, wenn auch geringfügiges, Problem aus Sicht der Barrierefreiheit stellen inaktive Formularfelder dar, wenn die »ausgegrauten« Informationen so kontrastarm sind, dass sie nicht mehr von allen Nutzern wahrgenommen werden können. Zwar nehmen die WCAG 2.0 solche Felder ausdrücklich von den Vorgaben für Mindestkontraste aus (»Für Text oder Bilder eines Textes, die Teil eines inaktiven Bestandteils der Benutzerschnittstelle [...] sind [...] gibt es keine Kontrastanforderung.«), trotzdem möchten viele Nutzer wissen, welche Funktionen zurzeit nicht zur Verfügung stehen – sie können diese aber nicht entziffern. Eine zusätzliche Kennzeichnung inaktiver Elemente, z.B. per Icon, würde das Interface unnötig verkomplizieren, daher bleibt hier nur die Variante, per CSS die generell etwas zu schwachen Kontraste solcher Felder und Kontrollelemente zu erhöhen – allerdings nur bis zu dem Punkt an dem sie noch von normalen, aktiven Feldern unterscheidbar sind.

Auf weitere Probleme in der Unterstützung solcher Felder durch verbreitete Screenreader weist der Artikel »Screen Reader Support For Disabled & Read Only Form Fields« hin.

Formulare zum Ausdrucken

Ein leider viel zu wenig beachtetes Anwendungsszenario von web-basierten Formularen ist der Ausdruck. Nutzer wollen vielleicht ihre gemachten Eingaben ausdrucken und archivieren, oder das leere Formular von Hand ausfüllen. Mit ein wenig CSS können Sie Ihr Formular so aufbereiten, dass es auch im Ausdruck eine gute Figur macht, ohne dass Sie dafür den HTML-Code verändern müssen.

Dazu müssen Sie nur Ihr Print-Style-Sheet (Sie haben doch ein Print-Style-Sheet, oder?) um ein paar Eigenschaften erweitern. Hier sollten Sie den Kontrollelementen ein Aussehen mitgeben, das für den Ausdruck geeigneter ist als die gewohnten Kontroll-, Eingabe- und Auswahlelemente im Webbrowser. Bei Textfeldern ist dies ausgesprochen simpel: statt der üblichen Pseudo-3D-Border definiert man im CSS kurzerhand:

Tipp: Wenn sie in Ihren Formularen Buttons vom Typ `input type="submit"` benutzen, dann sollten Sie im CSS nie das INPUT-Element stylen, sonst sehen Ihre Absenden-Buttons nachher aus wie Textfelder. Stattdessen sollten Sie Selektoren wie `input[type=text]` verwenden (die allerdings ältere Internet Explorer nicht verstehen) oder den Eingabeelementen Klassen zuweisen und diese per CSS formatieren. Die eleganteste Lösung ist jedoch, das BUTTON-Element zu benutzen.

```
textarea, input[type=text] {  
    border: none;  
    border-bottom: 2px #000 solid;  
}
```

Im Ausdruck ergibt dies:

Label: _____

Bei anderen Formularelementen wie Radiobuttons oder Checkboxes ist dies nicht ganz so einfach, allein schon weil man in CSS nicht für alle Browser runde Formen definieren kann, die für Radiobuttons nötig wären. Sie können diese allerdings simulieren, indem Sie die Kontrollelemente per Image Replacement durch eine entsprechende Grafik ersetzen. Ein weiterer Ansatz wäre, die Elemente mit CSS auszublenden und mit Generated Content durch Sonderzeichen wie zum Beispiel das Ballot Box-Zeichen aus Zapf Dingbats (Beispiele: ☉ ☒ ☐ ☒ ☒) zu ersetzen. Diese werden zwar, wie viele Unicode-Zeichen auch, im Screenreader nicht vorgelesen – allerdings ist es auch ausgesprochen unwahrscheinlich, dass sich ein vollblinder Screenreader-Nutzer ein für den Ausdruck vorgesehenes Formular vorlesen lässt.

4. Dynamik in Formularen: JavaScript & AJAX

In den Anfangstagen des Web waren die Auswirkungen nicht-barrierefreier Formulare noch nicht so gravierend: es gab nur eine sehr begrenzte Anzahl möglicher Formular-Elemente. Die Anzahl der Fehler, die ein Entwickler machen konnte, war schon deswegen eher überschaubar. Außerdem hatten Hilfsmittel wie Screenreader eingebaute Reparatur-Mechanismen, um die größten Schnitzer auszubügeln.

Dass diese relativ entspannten Zeiten vorbei sind, erkennt man alleine schon am Umfang der Literatur zu diesem Thema: Das Kapitel »Forms and interaction« aus Joe Clarks Buch »*Building Accessible Websites*« von 2002 hatte man in einer guten Stunde durch und war damit auf dem damals aktuellen Stand der Technik. Seitdem hat sich die Komplexität von web-basierten Formularanwendungen vervielfacht und somit auch die Dinge, die man als Entwickler berücksichtigen muss. Dass die Entwicklung barrierefreier Formulare immer komplexer geworden ist, erkennt man am Umfang der Dokumentation: während die WCAG 1 noch mit einigen Seiten (mehr schlecht als recht) erklärender Texte auskam, so erstrecken sich die Techniken zur Umsetzung der WCAG 2.0 ausgedruckt auf über fünfhundert Seiten (hier die Deutsche Übersetzung der Techniques).

Dynamische Inhalte, d.h. Inhalte, die sich mit oder ohne Einwirkung des Nutzers verändern, sind für viele Computer-Hilfsmittel ein echtes Problem. Schon wenn herkömmliche statische Webseiten mit Skripten angereichert werden, kommen Screenreader oder Lupenprogramme häufig aus dem Tritt. Wenn sich Dinge innerhalb einer Seite ändern und diese Änderung nicht im unmittelbaren Fokus des Nutzers bzw. seiner Anwendung ist, führt dies Regelmäßig zu Problemen in der Nutzbarkeit. Diese Schwierigkeiten sind jedoch im Kern nicht auf die Änderung *per Skript* zurückzuführen, denn die Hilfstechnologien haben vergleichbare Probleme, wenn die Änderungen *statisch*, d.h. durch das Neuladen der Seite zustande gekommen sind. Das Problem ist also nicht die Dynamik im Speziellen, sondern die Änderung im Allgemeinen.

Besonders problematisch aus Sicht der Nutzer von Hilfsmitteln sind Formulare, bei denen Inhalte dynamisch nachgeladen werden. Viele Werkzeuge und selbst modernste Screenreader haben je nach Art der Umsetzung Probleme damit, Veränderungen in einer Seite zu erkennen und dem Nutzer entsprechend zu präsentieren – teilweise werden Änderungen komplett ignoriert, oder sie werden wie ein Neuladen der Seite behandelt und der Screenreader beginnt wieder ganz oben auf der Seite.

Bei Desktop-Anwendungen können sich Nutzer oftmals damit behelfen, indem sie Ihre Hilfsmittel sehr genau auf das Verhalten der häufig benutzten Anwendungen einstellen. Der verbreitete Screenreader JAWS bietet zum Beispiel die Möglichkeit, Anwendungen mit sogenannten »JAWS Scripts« zugänglich zu machen. Mittlerweile ist um dieses Prinzip herum ein ganzes Biotop von Anbietern entstanden, die solche Skripte für alle möglichen Anwendungen, von Office-Paketen bis zu Skype und iTunes, entwickeln und kostengünstig oder gratis ins Netz stellen.

Für Anwendungen, die im Webbrowser laufen, gibt es diese Möglichkeiten (bisher) kaum. Hilfsmittel wie Screenreader, Lupenprogramme usw. sind in den letzten Jahren zwar stark verbessert worden und kommen nun, auch dank Techniken wie WAI-ARIA, eher mit Dynamik in Webseiten zurecht. Allerdings beziehen sich solche Aussagen immer nur auf die neuesten Versionen der Programme – man kann alleine wegen der teils erheblichen Kosten für Updates nicht davon ausgehen, dass mit dem Tag der Veröffentlichung einer neuen Version alle Nutzer sofort umsteigen und damit alle Probleme gelöst sind.

Daher gilt nach wie vor die Empfehlung, zuerst eine statische Variante eines Formularprozesses zu entwickeln, bei der die gesamte Logik zunächst nur auf dem Server läuft und die nicht zwingend auf clientseitiges JavaScript angewiesen ist. Dieses kommt erst im zweiten Schritt zum Zuge. Positiver Zusatznutzen: So bedienen Sie auch die je nach Meßmethode 1 bis 6% Nutzer, bei denen JavaScript bewusst abgeschaltet ist oder aus anderen Gründen nicht ausgeführt werden kann. Anbietern, die noch der alten BITV 1 unterliegen bleibt zudem nichts anderes übrig, da diese in Anforderung 6 verlangt:

»Internetangebote müssen auch dann nutzbar sein, wenn der verwendete Benutzeragent neuere Technologien nicht unterstützt oder diese deaktiviert sind.«

<zit>

JavaScript: vom optionalen Sahnehäubchen zur unverzichtbaren

Zutat

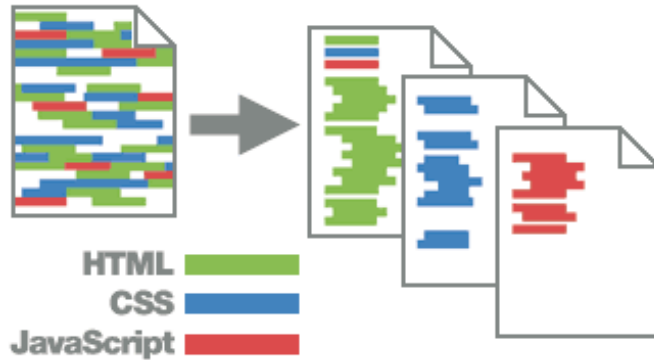
Nachdem JavaScript lange Zeit verpönt war und viele sinnvolle Anwendungen de facto verboten waren, hat diese clientseitige Skriptsprache in den letzten Jahren eine wahre Renaissance erlebt. Und zu Recht: richtig angewendet kann JavaScript gerade in komplexen Formularen oder web-basierten Anwendungen den Nutzer unterstützen, Fehlbedienungen verhindern und ihm eine Menge Arbeit und Wartezeit abnehmen. JavaScript kann damit gerade für Nutzer mit Behinderung Barrieren abbauen.

Mit JavaScript können Sie zum Beispiel Eingaben des Benutzers überprüfen, bevor er das Formular absendet. Ohne clientseitige Prüfung müsste er auf eine Antwort des Servers warten, der eine komplett neue Seite erstellen und ausliefern muss. Mögliche Anwendungszwecke für diese Technik sind zum Beispiel Trefferlisten oder Rechtschreibkorrekturen in Suchmaschinen und bei der Eingabe von Texten, Hinweise, dass bestimmte Kombinationen von Eingaben nicht möglich sind oder die sofortige Überprüfung von Eingaben mit hoher Fehlerquote. Bei einer Nutzerregistrierung können Sie Ihre Kunden schon während der Eingabe darauf hinweisen, dass ein gewünschter Benutzername bereits vergeben oder das gewählte Passwort zu unsicher ist und Vorschläge zur Korrektur machen.

All dies ist zwar auch ohne aktives Scripting möglich, dies bedeutet jedoch für den Nutzer eine längere Wartezeit und eine weniger unmittelbare Interaktion mit der Anwendung. Mit JavaScript können Sie Ihr Formular hingegen mit Komfortmerkmalen und unterstützenden Funktionen anreichern und so auf die Bedürfnisse von Nutzern mit kognitiven oder Lernbehinderungen reagieren.

Solche Skripte funktionieren jedoch nur, wenn mit der Zugangsart des Benutzers JavaScript tatsächlich ausgeführt werden kann. Für Fälle, in denen JavaScript ausgeschaltet ist oder nur unzureichend unterstützt wird, hilft nur ein sogenanntes »Fallback« auf den Server, der identische Funktionen übernimmt. Eine Verlagerung von Logik und Rechenleistung auf den Client sollten Sie daher immer erst dann vornehmen, wenn diese auch auf dem Server bereitgestellt wird.

In den guten alten Zeiten des Webdesigns waren die Fronten klar und die Zuständigkeiten ordentlich verteilt: auf dem Server lief die Geschäftslogik, HTML sorgte für strukturierte Inhalte und Funktionen, CSS lieferte die Oberfläche und obendrauf konnte man mit JavaScript noch ein wenig optionalen Komfort mitgeben. Wenn man in diesem Schichtenmodell die Verhaltensschicht wegnimmt, hat man nach wie vor eine ansehnliche und funktionale Seite. Nimmt man auch noch CSS weg, so bleibt das reine HTML der Seite. Nicht mehr ganz so hübsch, aber eine weiterhin funktionierende Struktur, bestehend aus dem absolut notwendigen Grundgerüst der Seite oder der Anwendung.



In Zeiten wo web-basierte Applikationen Aufgaben übernehmen, die früher reinen Desktop-Anwendungen vorbehalten waren, lässt sich diese strikte Trennung nicht mehr länger durchsetzen. Während Angebote wie Google Mail noch eine statische HTML-Alternative anbieten, die auf Kosten der Interaktivität ohne JavaScript auskommt, so ist dies bei der browser-basierten Tabellenkalkulation Google Text und Tabellen prinzipiell nicht mehr möglich. Wie viele der sogenannten Web2.0-Angebote beruht auch sie auf Funktionalitäten, die sich nicht mehr serverseitig nachbilden lassen. Ohne JavaScript-Unterstützung wären sie nur derart umständlich zu bedienen, dass sie niemand nutzen könnte. Der Erfolg solcher Angebote hängt also unmittelbar davon ab, dass die Browser der Anwender sämtliche Webstandards korrekt unterstützen – was im vorliegenden Fall ausdrücklich JavaScript mit einschließt.

So sollte man nicht mehr JavaScript pauschal für unzugänglich erklären; stattdessen ist die spannende Aufgabe herauszufinden, wie aktive und dynamische Inhalte mit den besonderen Bedürfnissen von Menschen mit Behinderung und den Fähigkeiten der Computer-Hilfsmittel in Einklang zu bringen sind. Auch hier hilft der Blick in die WCAG 2.0 – viele dieser Bedürfnisse erschließen sich besser, wenn man die Hintergründe für die einzelnen Prinzipien, Richtlinien, Erfolgskriterien und Techniken kennt.

Folgerichtig gehen die WCAG 2 dieses Thema nun endlich wesentlich entspannter an: statt JavaScript pauschal für unzugänglich zu erklären wird für den Einsatz lediglich die Bedingung aufgestellt, dass die Technik nachweislich mit verbreiteten Web Clients und Hilfsmitteln nutzbar ist:

»Die Nutzung einer Technik auf eine die Barrierefreiheit unterstützende Art und Weise bedeutet, dass sie mit assistierenden Techniken (AT) und den Barrierefreiheitsfunktionen von Betriebssystemen, Browsern und anderen Benutzeragenten funktioniert.«

<zitat>

Gängige Funktionen und ihre typischen Probleme

Betrachten wir einige typische Funktionen von Web-basierten Anwendungen:

Inhalte werden dynamisch verändert, mit oder ohne Zutun des Nutzers und ohne dass die Seite bzw. Anwendung komplett neu vom Server geladen wird.

- Für Nutzer assistierender Programme wie Screenreader und Lupensoftware besteht hier das Problem, dass die Programme Änderungen im Browser oft nicht bemerken und den Nutzer weder informieren noch zu den Änderungen hinführen können. Teilweise verhalten sich Hilfsmittel bei Veränderungen in einer Seite so, als sei die Seite komplett neu geladen worden und beginnen z.B. wieder ganz oben an vorzulesen.
- Auch für normalsichtige Nutzer können bei solchen Funktionen eines Web-Angebots Hürden entstehen. Zunächst müssen die geänderten Konzepte im Web 2.0 verstanden werden – im Gegensatz zu statischen (Text-) Seiten verlangen web-basierte Applikationen wesentlich umfassendere Fertigkeiten im Umgang mit Inhalten und Funktionen – für Nutzer mit Lernbehinderungen definitiv eine Barriere. Allerdings: eine statische Alternative scheidet oftmals aus grundsätzlichen Überlegungen aus – ein Börsenticker, der sich nicht selbsttätig aktualisiert ist ein Muster ohne Wert.
- Dabei entstehen Barrieren nicht nur bei der Verwendung von spezialisierten Hilfsmitteln – manche dynamische Techniken führen auch schon bei Nutzungsarten zu Problemen, die wesentlich häufiger anzutreffen sind. So führt der Trend zu endlos lang scrollenden Seiten wie z.B. bei Twitter dazu, dass

weite Teile des Interfaces per Tastatur schlichtweg nicht mehr erreichbar sind.



Für Twitter-Nutzer gibt es nun zum Glück mit Accessible Twitter ein alternatives Interface – allerdings wäre es aus Sicht der Betroffenen wünschenswerter, wenn das Original zugänglich wäre.

Statische Inhalte werden erst durch Interaktion zu dynamischen Elementen.

- Ein Beispiel für diese Technik findet sich in der Foto-Community flickr: hochgeladene Bilder werden zunächst mit den Bezeichnungen abgelegt, mit denen die Digitalkamera die Bilder benannt hat (DSC_1234.jpg); flickr verwendet diese für die korrekt als Überschriften (H2) ausgezeichneten Titel der Bilder. Klickt man diese Überschriften mit der Maus an, so werden diese per Skript in das Textinputfeld eines Formulars verändert. Hier kann man nun den Titel des Bildes ändern und diese Änderung speichern

oder verwerfen:



Kathedrale

SAVE
CANCEL

- Eigentlich eine gute Idee. Das Problem bei dieser Form der Interaktion beginnt jedoch bereits mit der Tatsache, dass diese Überschriften in den meisten Browsern nicht per Tastatur erreichbar sind. Es sei denn man verwendet das für Überschriften (h1-6) eigentlich nicht erlaubte `tabindex`-Attribut und riskiert so, dass der Code nicht mehr validiert. Hinzu kommt das Problem der »Entdeckbarkeit« (in der englischsprachigen Fachliteratur »*Discoverability*« genannt): welches Element einer Seite tatsächlich auf diese Art editierbar ist erfährt der Nutzer erst durch Ausprobieren.

Anwendungen bilden Funktionen nach, die in HTML (bisher) nicht vorgesehen sind.

- Für einfache Zwecke reicht der begrenzte Satz von Formularelementen, den HTML bietet, in der Regel vollkommen aus. In web-basierten Anwendungen stößt man jedoch schnell an die Grenzen dieser Markup-Sprache, die nie für solche Zwecke gedacht war. Man sehnt sich nach einer schnelleren Standardisierung von HTML5 oder XForms. Bis dahin bleibt dem Webentwickler nichts anderes übrig, als Funktionen bzw. Elemente wie Fortschrittsbalken, Schieberegler, Tri-State-Checkboxes u.v.m. mit den vorhandenen Mitteln nachzubilden.
- Und genau hier beginnt das Problem für viele Nutzungsszenarien, die für Menschen mit Behinderungen typisch sind: Viele dieser Elemente sind zum Beispiel nicht per Tastatur zu bedienen, da es sich nicht um echte Interface-Elemente handelt, sondern beispielsweise nur um leere DIV und SPAN, die per Maus und JavaScript verschoben werden.
- Einerseits haben wir also das Problem der Geräteunabhängigkeit in der Bedienung, und darüber hinaus sind Hilfsmittel auch noch darauf angewiesen, dass es in der jeweiligen Accessibility-API des Betriebssystems ein entsprechendes Pendant zu den Formular-

Weitere Links zum Thema:

- AOL Developer Network: »AXS Library«
- Hans Hillen: »jQuery Widget Samples«
- Jens Grochtdreis: »Klappfunktion mit jQuery und WAI-ARIA accessible machen.«
- Dirk Ginader: »Accessible Tabs«
- Dirk Ginader: »Accessible Carousel«
- Dirk Ginader: »LogFocus – handy Bookmarklet for Keyboard Accessibility testing«
- Aaron Gustafson: »TabIndex in Firefox using NVDA screenreader«

und Kontrollelementen gibt, um die Bedienbarkeit zu gewährleisten. So wird z.B. ein Fortschrittsbalken in echten Desktop-Anwendungen – wenn es bei Uploads, komplexen Abfragen oder Berechnungen mal wieder etwas länger dauert – im Screenreader erkannt; die Werte werden in Intervallen wiedergegeben (20%, 30%, 40%, ...). Bildet man diese Funktion (zumindest optisch) mit den Bordmitteln von HTML4, CSS & JavaScript nach, so mag dies zwar wie ein Fortschrittsbalken aussehen, für einen Screenreader ist dies jedoch eine Ansammlung sinnfreier leerer Elemente. Hier werden Sie also für die Herstellung der Barrierefreiheit nicht umhin kommen, entweder auf HTML5 und seine erweiterten Möglichkeiten zu setzen, oder die Funktionen bei einer älteren Codebasis (HTML4/XHTML1) per WAI-ARIA nachzurüsten.

■ CodeTalks: »Set of ARIA Test Cases«

Ein Praxisbeispiel: Formularbereiche ein- und ausblenden

Bei komplexen Formularen kann es sinnvoll sein, nur die im Kontext eines aktuellen Arbeitsschrittes relevanten Bereiche eines Formulars anzuzeigen. Banken und Finanzinstitute z.B. haben oft Formulare mit einer Reihe von Variablen, und welche Daten der Nutzer eingeben muss hängt unmittelbar davon ab, welche Variable der Nutzer auswählt: möchte er Geld überweisen oder einen Kredit beantragen? Oder will er sein Auto oder sein Haus versichern?

Aus Sicht der allgemeinen Gebrauchstauglichkeit scheint es sinnvoll, dem Nutzer nur die Teile der Formulare zu zeigen, die im vorliegenden Fall relevant sind. Dies kann insbesondere für Nutzer mit kognitiven Behinderungen oder Lernschwierigkeiten hilfreich sein, aber auch für Nutzer mit wenig Computer-Erfahrung.

Nun kann man eine Kombination aus HTML, CSS und JavaScript verwenden, um verschiedene Bereiche eines Formulars ein- und auszublenden. Gerade hier sollte man sehr sorgfältig vorgehen um sicherzustellen, dass neu eingeblendete Bereiche zum Beispiel für Screenreader-Nutzer zugänglich sind. Gleichzeitig sollte verhindert werden, dass das Programm bei einer Änderung des Inhalts wieder zum Beginn der Seite springt.

Hier hilft die Lektüre der Dokumentation zu den WCAG 2.0: die Technik SCR26 wird empfohlen, um das Erfolgskriterium 2.4.3 – Fokus-Reihenfolge – zu erfüllen:

»SCR26: Einfügen von dynamischen Inhalten in das Document Object Model unmittelbar anknüpfend an sein auslösendes Element«

< zitat >

Das W3C erklärt die Technik wie folgt:

»Diese Technik fügt neuen Inhalt in das DOM unmittelbar nach dem Element, das aktiviert wurde, um das Skript auszulösen, ein. Das auslösende Element muss ein Link oder eine Schaltfläche sein und das Skript muss von seinem onclick-Event aus aufgerufen werden. Diese Elemente sind nativ fokussierbar und ihr onclick-Event ist geräte-unabhängig. Der Fokus bleibt auf dem aktivierten Element und der neue Inhalt, dahinter eingefügt, wird sowohl in der Tabreihenfolge als auch in der Lesereihenfolge der Screenreader der nächste Punkt.

< zitat >

Beachten Sie, dass diese Technik bei synchronen Aktualisierungen funktioniert. Bei asynchronen Aktualisierungen (manchmal AJAX genannt) wird eine zusätzliche Technik benötigt, um die assistierende Technik darüber zu informieren, dass der asynchrone Inhalt eingefügt wurde.«

Das folgende Formular enthält einen sich verändernden Bereich zur Auswahl einer Eissorte, der je nach Auswahl der Radiobuttons mit unterschiedlichen Inhalten befüllt wird. Wenn der Radiobutton »Hörnchen« ausgewählt wird, bekommt der Nutzer die Abfrage der Größe zusätzlich zur Geschmacksrichtung, bei der Auswahl »Becher« entfällt die Größenabfrage. Der Code in der Beispieldatei eissorten.html ist in aktuellen Versionen der Screenreader JAWS und Window Eyes getestet:

Noch ein Praxisbeispiel: Auswahloptionen ändern

Manche Formulare enthalten eine Reihe von Auswahlmenüs auf Basis des SELECT-Elements, und je nach Auswahl ändert sich der Inhalt weiterer Formularelemente. Dieses Szenario wird unter anderem in Richtlinie 3.2 der WCAG 2.0 behandelt, wo es heißt:

»3.2.5 Änderung auf Anfrage: Änderungen des Kontextes werden nur durch Benutzeranfrage ausgelöst oder es gibt einen Mechanismus, um solche Änderungen abzuschalten.«

Das WCAG 2.0 Erfolgskriterium 3.2.5 deckt eine ganze Reihe von möglichen Barrieren ab und listet Techniken zu ihrer Vermeidung auf. Im Zusammenhang mit der Benutzung solcher SELECT-Menüs wird die folgende Technik empfohlen:

»SCR19: Benutzung eines onchange-Events auf einem ausgewählten Element, ohne eine Änderung des Kontextes auszulösen«

SCR19

Vergleichbare Vorgaben finden sich auch im Prüfverfahren des BIENE-Wettbewerbs:

»18. Änderungen von Teilbereichen einer Bildschirmseite werden sinnvoll eingesetzt, angekündigt und sind durch die Nutzerin / den Nutzer kontrollierbar.«

Kriterium 18

»19. Jegliche Funktion der Seite ist auch über die alleinige Verwendung der Tastatur in einer schlüssigen Reihenfolge zu erreichen, wobei die jeweils ausgewählte Funktion in der Standardansicht gut sichtbar ist.«

Kriterium 19

Das folgende Beispiel enthält zwei SELECT-Elemente: »Kontinent« und »Land«. Wenn ein Kontinent ausgewählt wird, z.B. »Nordamerika«, dann werden die möglichen Länder für diesen Kontinent eingeblendet (Kanada, Mexiko & USA). Wenn ein anderer Kontinent ausgewählt wird, dann wird die folgende Länderauswahl automatisch geändert und stellt nun die eine andere Liste von Ländern zur Auswahl (Deutschland, Frankreich, Italien, ...).

SELECT-Elemente dynamisch befüllen

Kontinent:

Land:

JavaScript wird hier benutzt, um die angebotenen Auswahlmöglichkeiten im zweiten SELECT auf Basis der Auswahl im ersten SELECT zu verändern. Der Code in der Beispieldatei laenderauswahl.html mit dem benötigten HTML, CSS und JavaScript ist in aktuellen Versionen von JAWS und Window Eyes getestet (Beispiele adaptiert mit freundlicher Erlaubnis von Roger Hudson). Der Vollständigkeit halber sollten Sie dann noch Vorkehrungen für das Szenario treffen, dass JavaScript nicht ausgeführt werden kann, z.B. durch ein statisches Fallback mit allen verfügbaren Optionen.

Theorie der Praxis und Praxis der Theorie

Anders als bei klassischen Webseiten, die sich an der statischen Dokumenten-Metapher orientieren, lassen sich Barrieren in web-basierten Anwendungen nur schwer vorhersagen und somit auch kaum in ein abstraktes Regelwerk für systematische Tests einsortieren. Selbst wenn Sie für Ihre Anwendung eine JavaScript-Library wie Dojo verwenden, bei der die Accessibility eine wichtige Rolle spielt, ist noch lange nicht garantiert, dass das Ergebnis auch tatsächlich barrierefrei zu nutzen ist.

Sie werden also nicht umhin kommen, Ihre Formulare einem Praxistest durch Nutzer verschiedener Hilfsmittel zu unterziehen. Gerade wenn man moderne Techniken wie HTML5, WAI-ARIA und unaufdringliches JavaScript einsetzt wird es auch in den neuesten Versionen der Hilfsmittel zu Fehlern kommen. Um ein konkretes Beispiel zu geben: Sie können sich nicht darauf verlassen, dass Ihr Code in allen möglichen Browser- & Hilfsmittel-Konfigurationen wie gewünscht funktioniert. Es gibt eine ganze Reihe empfehlenswerter Techniken, die zum Beispiel in JAWS 10 zusammen mit Internet Explorer 8 ohne negativen Befund einen Test passieren, nimmt man JAWS 10 zusammen mit Firefox, dann funktioniert es auf einmal nicht mehr (und auch der umgekehrte Fall ist möglich). Eine gute Hilfe ist in solchen Fällen (neben Tests mit echten Nutzern) die Testreihe »Set of ARIA Test Cases« bei CodeTalks.org – hier finden Sie erwartete und tatsächlich eingetretene Testergebnisse für typische Anwendungsfälle von Web-basierten Widgets.

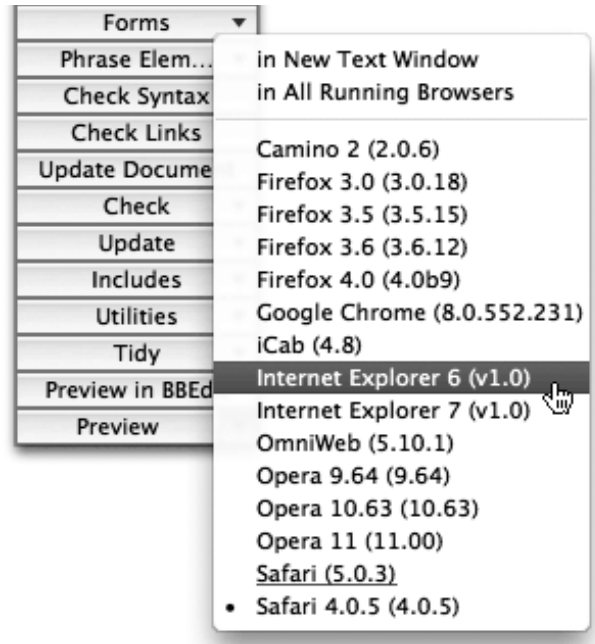
Um die Barrierefreiheit wirklich sicherzustellen dreht sich in der nächsten Folge alles um das »Testen von Formularen und web-basierten Anwendungen«.

5. Testen von Formularen und web-basierten

Anwendungen

Klassische Webseiten, also web-basierte Text-Dokumente, sind relativ einfach zu testen. Was man machen darf und was nicht ist genau beschrieben, mögliche Fehler sind vorhersehbar und somit vermeidbar. Die Vermeidung von technischen Barrieren gehört mittlerweile zum Handwerkszeug eines guten Frontend-Entwicklers; Konzepter und Grafiker orientieren sich an Bedürfnissen und Gewohnheiten der Kunden. Jeder gute Redakteur ist daran interessiert, seine Leserschaft nicht mit seinen Texten zu überfordern. Eine ganze Reihe der Kriterien zur Barrierefreiheit sind sogar vollständig maschinell testbar oder zumindest halbautomatisch durch geschulte Tester zu verifizieren.

Bereits bei einfachen Formularen oder Mischformen aus Dokumenten- und Formular- bzw. Anwendungstypen stimmt diese Aussage jedoch nicht mehr: hier ist man noch stärker auf die Beobachtung echter Nutzer angewiesen. Technische Hürden lassen sich zwar auch im Web 2.0 mit maschineller Unterstützung testen; so bietet die YUI Library sogar ein eigenes Test-Framework, das JavaScript-basierte Web-Anwendungen gegen die Implementierungen in allen halbwegs standardkonformen Browsern testet. Solche Tests können jedoch nur verhindern, dass man per JavaScript den Browser des Nutzers »abschiesst«. was tatsächlich in Screenreader & Co. ankommt können Sie nur mit echten Nutzern solcher Programme herausfinden.



Dabei kann man nur dringend davon abraten, als Entwickler mal eben so im Vorbeigehen mit einem

Screenreader zu testen. Zum Einen läuft man so Gefahr, das Thema Accessibility einzig und allein auf den fast schon sprichwörtlichen »blinden Nutzer mit JAWS« zu reduzieren; zum Anderen sind solche Hilfsmittel schon fast Expertensysteme, bei denen man wochenlange Einarbeitung braucht, um zumindest die nötigsten Tastaturbefehle verinnerlicht zu haben. Ein salopper Test bringt daher im Zweifelsfall kaum brauchbare oder im Ernstfall sogar falsch positive Ergebnisse.

Zu einem vollständigen Test gehört, dass die Formulare nicht nur in ein oder zwei gängigen Browsern visuell überprüft werden, auch alternative Ein- und Ausgabeformen sollten berücksichtigt werden. Da es sich bei vielen assistiven Programmen um teils sehr komplexe Anwendungen handelt sollten Sie im Idealfall immer mit echten Nutzern solcher Werkzeuge testen.

Aber auch bereits während der Entwicklung eines komplexen Formulars kann es erhellend sein, Zwischenstände mit den entsprechenden Programmen zu testen. Bei der Initiative WebAIM gibt es eine Reihe Anleitungen, wie man verbreitete Screenreader zur Evaluation der Barrierefreiheit einsetzen kann:

- »Using JAWS to Evaluate Web Accessibility«
- »Using NVDA to Evaluate Web Accessibility«

Testen, testen, testen und immer an den Nutzer denken ...

Die Tests, die Sie bereits während der Entwicklung eines Angebots selber durchführen können sind meist recht simpel und bedürfen nicht mehr als ein paar unterschiedlicher Browser und diverser Erweiterungen.

In unserer »BITV-Reloaded«-Serie hatten wir für die dort beschriebenen Tests die »Web Developer Toolbar« für Firefox empfohlen; bei diesen Tests können Sie sich hingegen von der »Firefox Accessibility Extension« der University of Illinois unterstützen lassen. Aktuelle Versionen dieser Erweiterung unterstützen bereits den kommenden WAI-ARIA-Standard (*Accessible Rich Internet Applications*) und sind somit gerade für das Testen von web-basierten Anwendungen interessant. Neben den Optionen zum An- und Abschalten bestimmter Elemente lassen sich mit dieser Erweiterung auch selektiv bestimmte JavaScript-Events unterdrücken. Zudem erfährt man eine Menge über verwendete Events sowie Rollen und Zustände von Bedienelementen und Widgets im Sinne der WAI-ARIA.

Der Minimaltest

Für eine detaillierte Beschreibung von Barrieren und wie man sie aufspürt, legen wir Ihnen nochmals unsere »BITV-Reloaded«-Serie und das Prüfverfahren zum BIENE-Wettbewerb nahe. An dieser

Weitere Links zum Thema:

Stelle nun einige Tipps, wie Sie Ihre Formulare mit ein paar kurzen Tests überprüfen können:

Sind alle Inhalte und Funktionen per Tastatur erreichbar und bedienbar?

Legen Sie die Maus beiseite und bewegen Sie sich ausschließlich per Tastatur durch Ihr Angebot. Im Gegensatz zu reinen Textseiten, wo die Navigation per Tastatur je nach Browser nur mit einem begrenzten Satz von Tasten funktioniert und Formularelemente automatisch auch per Tastendruck zugänglich sind, können in web-basierten Applikationen weitere Tasten wie Leertaste, Escape, Control, Alt und Command mit Funktionen belegt sein. Achten Sie bei Ihren Tests auch darauf, dass diese nicht nur funktionieren, sondern auch irgendwo erklärt sind!

- Henny Swan: »Establishing a screen reader test plan – Accessibility testing«
- Jim Thatcher: »Favelets for Checking Accessibility«

Sind alle Label korrekt verknüpft?

Für diesen Test dürfen Sie die Maus nun wieder zur Hand nehmen: klicken Sie auf sämtliche Labels Ihrer Formulare. Achten Sie besonders bei Radiobuttons und Checkboxes darauf, dass die jeweils korrekten Elemente markiert werden und nicht irgendwelche anderen – dann haben Sie `id`- oder `for`-Attribute falsch gesetzt und müssen diese korrigieren. Ein Nutzer, der auf Tastaturbedienung angewiesen ist, hat keine Chance, ein nicht korrekt zugeordnetes Element über sein Label zu bedienen – ein Fehler, den kein Validator bemerken kann und den Sie nur durch tatsächliches Ausprobieren finden können.

Haben alle Radiobuttons einer Gruppe das gleiche name-Attribut?

Ein weiterer Fehler, den Sie nicht mit einem Validator finden können, sondern nur durch Ausprobieren: Damit Radiobuttons sich wirklich gegenseitig ausschließen, müssen alle zusammengehörigen Radiobuttons den gleichen Wert im `name`-Attribut stehen haben, ansonsten ist es möglich mehrere Buttons einer entweder-oder-Auswahl zu drücken.

Werden durch Navigationsbewegungen Formulare ungewollt verändert oder sogar abgeschickt?

Ein häufiger Fehler ist, Formularelemente mit JavaScript per `onchange` oder `onblur` zu prüfen. Diese Art der Überprüfung ist zwar direkt und schnell und damit scheinbar hilfreich, kann aber das Ausfüllen eines Formulars per Tastatur oder Spracherkennung schlichtweg unmöglich machen. Eine detailliertere Diskussion dieser Barrieren finden Sie in unserer »BITV Reloaded«-Serie bei Anforderung 9, Bedingung 9.3

Die wohl am meisten verbreitete Anwendung dieser Technik sind Drop-down-Menüs, die gerne genommen werden, um Platz zu sparen und die Navigation bzw. die Anzahl der Optionen weniger komplex erscheinen zu lassen. In den weitaus meisten Fällen wäre hier ein klassischer Hyperlink das semantisch korrektere Element, da es sich streng genommen ja auch nur um Hyperlinks handelt. Hyperlinks haben außerdem den Vorteil, dass sie nur einen Klick bzw. Tastendruck benötigen. Eine Liste von Hyperlinks ist zudem einfacher zu erfassen. Drop-down-Menüs bedeuten für den Nutzer mehr Lernaufwand und Mühe in der Bedienung: ein Drop-down-Menü muss man ansteuern und aufklappen, um alle Optionen zu sehen. Selbst dann sieht man je nach Anzahl der Optionen oder je nach Browser nicht alle Optionen auf einen Blick und muss die Liste herunterscrollen.

Bei Ihren Tests sollten Sie also darauf achten, dass Funktionen, die letztendlich nichts anderes als Hyperlinks sind, auch als Hyperlinks umgesetzt wurden.

Ist alles auch ohne Farbe erkennbar?

Achten Sie insbesondere bei Fehlermeldungen darauf, dass diese nicht alleine durch farbliche Markierungen gekennzeichnet sind. Wenn Sie beim Testen auf Fehlermeldungen durch den unvollständigen Versand eines Formulars stoßen, dann benötigen diese zusätzliche Struktur- und Gestaltungselemente jenseits von roter Hintergrundfarbe, um wirklich in allen Ausgabeformen wahrnehmbar und damit nutzbar zu sein. Gleiches gilt sinngemäß nicht nur für die Meldung, sondern auch für die Markierung des Ortes des Fehlers.

Für manche Formen der Sehbehinderung bietet das Windows-Betriebssystem so genannte Kontrastmodi. Typischerweise wird durch diese der Bildschirminhalt invertiert bzw. mit benutzerdefinierten Farben dargestellt, also zum Beispiel beiger Text auf schwarzem Hintergrund, um eine Blendwirkung zu vermindern. Diese Kontrastmodi haben aus Designer-Sicht eine unangenehme Eigenschaft: während alle

CSS-Positionierungen beibehalten werden, verschwinden sämtliche Hintergrundfarben und vor allem auch Hintergrundbilder! Wenn Sie also Warnhinweise per CSS (z.B. `.fehler {background-image: foo.gif;}`) realisiert haben, so sind diese für viele sehbehinderte Nutzer nicht mehr wahrnehmbar, weil schlichtweg nicht vorhanden. Um das zu vermeiden, sollten Sie Ihre gesamte Anwendung unbedingt einmal vollständig im Windows-Kontrastmodus durchtesten.

Gibt es eine logische Abfolge der Elemente?

Für viele Nutzungsszenarien ist es von immenser Bedeutung, dass die Abfolge in einem Formular sowohl der optischen Anordnung als auch der Reihenfolge im Quelltext entspricht. Um das zu prüfen benötigen Sie gegebenenfalls eine Browser-Erweiterung wie zum Beispiel die Web Developer Toolbar für Firefox oder WAVE, mit denen Sie die Style Sheets ihrer Seite oder Anwendung unterdrücken können. Schalten Sie CSS ab und testen Sie, ob die Formulare immer noch bedienbar sind.

Zum Schluss: eine kurze Checkliste

Das Thema Barrierefreiheit lässt sich nicht auf eine simple Checkliste reduzieren – dafür ist es einfach zu komplex, und letztendlich geht es ja um Menschen, nicht um Maschinen. Trotzdem hilft eine Checkliste gerade bei der Erstellung komplexer web-basierter Anwendungen, weil man ja immer mal etwas vergessen kann. Daher hier eine kurze Checkliste (ohne Anspruch auf Vollständigkeit):

- Entfernen Sie unnötige Inhalte von den Seiten
- Verlangen Sie nur wirklich benötigte Daten
- Akzeptieren Sie unterschiedlichste Datenformate und Schreibweisen
- Achten Sie auf die Tastatur-Bedienbarkeit
- Erstellen Sie eine logische Tab-Reihenfolge
- Achten Sie auf eine ruhige Gestaltung – gleiche Dinge sollten gleich aussehen und sich gleich verhalten.
- Kennzeichnen Sie Pflichtfelder
- Verwenden Sie sinnvolle Feldbeschriftungen
- Sprechen Sie den Nutzer aktiv an (z.B. durch Verben, die eine Aktion beschreiben)
- Übertreiben Sie es nicht bei der Anzahl auswählbarer Optionen
- Stellen Sie intelligente Vorauswahlen zur Verfügung
- Geben Sie dem Nutzer Auskunft über den Fortschritt innerhalb eines mehrteiligen Prozesses
- Verwenden Sie Reset- oder Abbrechen-Buttons nur in begründeten Ausnahmefällen
- Stellen Sie Informationen zum Datenschutz zur Verfügung
- Verifizieren Sie die eingegebenen Informationen client- und serverseitig
- Stellen Sie eventuell benötigte Anleitungen vor die betreffenden Felder
- Geben Sie bei Bedarf Hilfen oder stellen Sie bei komplexen Angeboten eine Hotline zur Verfügung
- Formulieren Sie freundliche und hilfreiche Fehlermeldungen
- Machen Sie Ort und Art des Fehlers deutlich
- Beschreiben Sie Möglichkeiten zur Problembehebung
- Auch bei fehlerhaften Eingaben müssen die korrekt eingegebenen Daten erhalten bleiben
- Vergessen Sie nicht die abschließende Bestätigungsseite
- Testen Sie Ihr Formular, indem Sie alle Labels nochmal anklicken (dies wird immer wieder vergessen, und ein Validator findet falsch zugeordnete Labels nicht)
- Überprüfen Sie Ihr Angebot mit Nutzertests, bevor es online geht

Weitere Kriterien zur Überprüfung Ihrer Website finden Sie in den Prüfschritten des BIENE-Wettbewerbs

Was sagen die WCAG 2.0 zu Formularen?

Die im Oktober 2009 als offizielle deutsche Übersetzung veröffentlichten Web Content Accessibility Guidelines (WCAG) 2.0 behandeln HTML-Formulare nicht direkt, da sie unabhängig vom Einsatzzweck und den verwendeten Techniken formuliert sind. Wertvolle Hinweise zur Umsetzung und Überprüfung finden sich

jedoch in den sogenannten »Techniken und Fehler für die Richtlinien für barrierefreie Webinhalte«, die mittlerweile ebenfalls ins Deutsche übersetzt wurden.

Zur Erfüllung der Vorgaben der WCAG 2.0 ist zu beachten, dass je nach angestrebter Konformitätsstufe unterschiedlich hohe Messlatten anzusetzen sind. Eine gute Erklärung hierfür gibt es in Folge 158 des Technikwürze-Podcasts:

»... WCAG 2 unterscheidet die Fehlerbehandlung in drei Stufen: Auf Level A (Fehlererkennung) muss ein Fehler zumindest erkennbar sein mit Hilfe von Semantik, Text und / oder Farbe. Auf Level AA (Fehlererläuterung) sollen die Fehler nicht nur erkennbar, sondern auch mit Beispielen und Hilfen etwa nahe am Formularfeld erläutert werden. Auf Level AAA (Fehlerprävention) liegen dann erst jene Schritte und Bestätigungen, die für uns bei ausführlicheren Formularprozessen ohnehin längst gängige Praxis sind. Formulare, die in Schritten durchlaufen werden, müssen auch in Schritten erkennbar und benutzbar sein und Bestätigungsseiten anbieten, die dem Nutzer entweder noch einmal eine Möglichkeit geben, die Inhalte zu bestätigen, oder schlicht die Ergebnisse des Formulars noch einmal klar anzeigen. Folgerichtig wird hierbei der Level bei finanziellen und rechtlichen Transaktionen dann auf AA gehoben.«

<zitat>

Zum Abschluss noch eine Liste mit wertvollen Hinweisen und Tipps aus den Richtlinien zum Thema Barrierefreie Formulare:

Die WCAG 2.0 zum Thema Label:

- H44: Benutzung von Label-Elementen, um Text-Label mit den Steuerelementen eines Formulars zu assoziieren
- H65: Benutzung des title-Attributs, um Formular-Steuerelemente zu kennzeichnen, wenn das Label-Element nicht benutzt werden kann

Die WCAG 2.0 zum Thema Gruppierung von Formularelementen:

- H71: Bereitstellung einer Beschreibung für Gruppen von Formular-Steuerelementen, indem fieldset- und legend-Elemente benutzt werden
- H85: Benutzung von OPTGROUP, um OPTION-Elemente innerhalb eines SELECT zu gruppieren

Die WCAG 2.0 zum Thema Fehlervermeidung und -behebung:

- G83: Bereitstellung von Textbeschreibungen, um nicht ausgefüllte Pflichtfelder zu identifizieren
- G85: Bereitstellung einer Textbeschreibung, wenn die Eingaben des Benutzers außerhalb des verlangten Formats oder der verlangten Werte sind
- G139: Erstellung eines Mechanismus, der es Benutzern erlaubt, zu den Fehlern zu springen
- SCR18: Bereitstellung einer client-seitigen Gültigkeitsprüfung und einer Warnmeldung
- SCR21: Benutzung von Funktionen des Document Object Model (DOM), um einer Seite Inhalte hinzuzufügen

Die WCAG 2.0 zum Thema Dynamik:

- SCR19: Benutzung eines onchange-Events auf einem ausgewählten Element, ohne eine Änderung des Kontextes auszulösen
- SCR26: Einfügen von dynamischen Inhalten in das Document Object Model unmittelbar anknüpfend an sein auslösendes Element

Die WCAG 2.0 zum Thema Zeitbeschränkungen:

- G133: Bereitstellung einer Checkbox auf der ersten Seite eines mehrteiligen Formulars, die es den Benutzern ermöglicht, um längere zeitliche Begrenzungen oder keine zeitliche Begrenzungen bei einer

Sitzung zu bitten

- SCR1: Es dem Benutzer ermöglichen, die Standardeinstellung der zeitlichen Begrenzung zu erweitern
- SCR16: Bereitstellung eines Scriptes, das den Benutzer warnt, dass eine zeitliche Begrenzung bald abläuft

Die WCAG 2.0 zum Thema Pflichtfelder:

- G14: Stellen Sie sicher, dass Informationen, die durch Farbunterschiede vermittelt werden, auch in Textform erhältlich sind
- G122: Einbeziehung eines Texthinweises immer dann, wenn Farbhinweise benutzt werden

Die WCAG 2.0 zum Thema Farben & Schriften:

- G18: Sicherstellen, dass es ein Kontrastverhältnis von mindestens 4,5:1 zwischen Text (und Bildern von Text) und dem Hintergrund hinter dem Text gibt
- G145: Sicherstellen, dass es ein Kontrastverhältnis von mindestens 3:1 zwischen Text (und Bildern von Text) und dem Hintergrund hinter dem Text gibt
- G148: Hintergrundfarbe nicht festlegen, Textfarbe nicht festlegen und keine Technikfunktionen benutzen, die diese Voreinstellungen ändern

6. Literatur

Bücher

- Michael Jendryschik (2008) »Einführung in XHTML, CSS und Webdesign – Standardkonforme, moderne und barrierefreie Websites erstellen« Addison-Wesley
- Peter Müller (2007) »Little Boxes« Markt + Technik (Teil 1)
- Dirk Jesse (2008) »CSS-Layouts – Praxislösungen mit YAML 3.0« Galileo Press
- Kai Laborenz (2008) »CSS-Praxis – Das umfassende Handbuch.« Galileo Press
- Shawn Lawton Henry (2007) »Just Ask: Integrating Accessibility Throughout Design« Lulu.com (Großdruck, Online-Version)
- Luke Wroblewski (2008) »Web Form Design – Filling in the Blanks« Rosenfeld Media
- Marti Hearst (2009) »Search User Interfaces« Cambridge University Press
- Caroline Jarrett, Gerry Gaffney (2008). »Forms that work: Designing web forms for usability« Morgan Kaufmann
- Jim Thatcher, Michael R. Burks, Christian Heilmann et al. (2006). »Web Accessibility: Web Standards and Regulatory Compliance« Friends of ED
- Matthew Linderman, Jason Fried (2004) »Defensive Design for the Web – How to Improve Error Messages, Help, Forms, and Other Crisis Points« New Riders
- John M. Carroll, Judith Reitman Olson (1987) »Mental Models in Human-Computer Interaction: Research Issues About What the User of Software Knows« National Academies Press
- Marti Hearst (2009) »Search User Interfaces« Cambridge University Press
- Derek Featherstone, Tim Connell, Jina Bolton (2008) »Fancy Form Design« Sitepoint
- Jan Eric Hellbusch, Kerstin Probiesch (2011) »Barrierefreiheit verstehen und umsetzen – Webstandards für ein zugängliches und nutzbares Internet« dpunkt

Tutorials & Ressourcensammlungen

- IBM Human Ability and Accessibility Center: »Developer guidelines | Web checklist | Checkpoint 1.3d: Forms«
- Dey Alexander: »Form design Discussion articles«

- Jessica Enders: »Articles – Formulate Information Design«
- Sarah Horton: »Universal Usability – A universal design approach to web usability«, Kapitel »Forms«
- Jim Thatcher: »Accessible Forms«
- Vitaly Friedman: »Web Form Design: Modern Solutions and Creative Ideas«
- Joe Clark: »Forms and interaction«
- ui-patterns.com: »User Interface Design patterns«
- Mark Pilgrim: »Dive Into Accessibility«