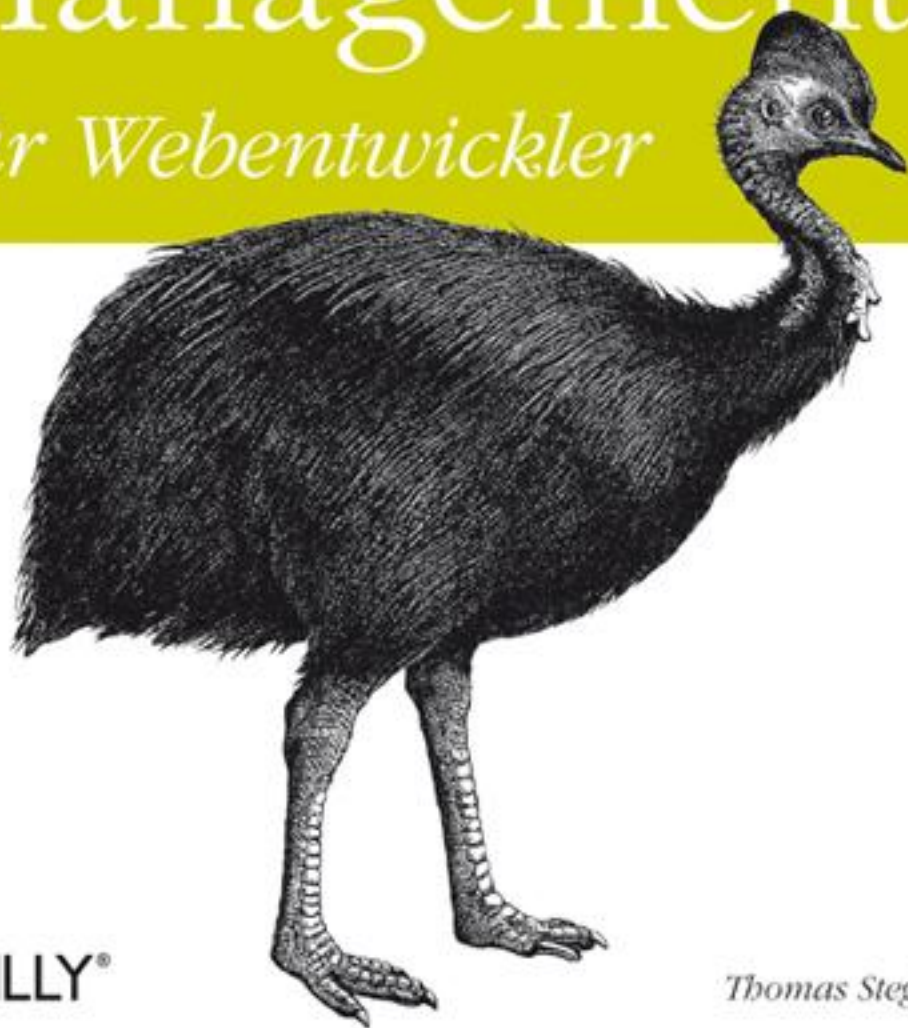


Techniken, Strategien, Beispiele

Deutsche
Originalausgabe

Zeit Management *für Webentwickler*



O'REILLY®

Thomas Steglich

| | |
|----------------------------------------------------|-----------|
| Einleitung | XI |
| 1 Zu viel Arbeit, zu wenig Zeit | 1 |
| Zeitdruck | 2 |
| Unterbrechungen | 2 |
| Multitasking | 3 |
| Prioritäten | 3 |
| Individuelle produktive Phasen | 4 |
| Gelungenes Zeitmanagement | 5 |
| 2 Aufgaben- und Terminverwaltung | 7 |
| Den Dingen auf den Grund gehen | 8 |
| Die »Getting Things Done«-Methode | 9 |
| Worum handelt es sich? | 10 |
| Was ist der nächste Schritt? | 15 |
| Was mache ich wann? | 17 |
| Den eigenen Zeitbedarf analysieren | 18 |
| Prioritäten setzen | 23 |
| Mit System flexibel bleiben | 27 |
| Revision – Ihr ganz persönlicher Systemcheck | 31 |
| Zeitmanagement mit PDA, PIM, PAA & Co. | 32 |
| Aufgaben | 47 |
| 3 Akquise und Marketing | 49 |
| Professionelles Auftreten | 50 |
| Corporate Design – Bekennen Sie Farbe | 50 |
| Kunden finden | 55 |

| | |
|-----------------------------------------------------|------------|
| Wege zum Kunden | 59 |
| Strategien zur Kundengewinnung | 61 |
| Customer Relationship Management (CRM) | 68 |
| Kundeninformationen nutzen | 73 |
| Aufgaben | 75 |
| 4 Projektplanung | 77 |
| Projektplanung anhand eines Beispielprojekts | 78 |
| Klassisches oder agiles Projektmanagement? | 94 |
| Wenn es eng wird | 98 |
| Werkzeuge für die Projektverwaltung | 101 |
| Dokumente im Workflow | 104 |
| Werkzeuge für Dokumente | 107 |
| Alternative Szenarien | 111 |
| Aufgaben | 114 |
| 5 Recherche und Analyse | 117 |
| Internetrecherchen | 118 |
| Das persönliche Gespräch | 124 |
| Zeitschriften | 125 |
| Fachbücher | 127 |
| Stoffsammlung | 129 |
| Weiterbildung | 134 |
| Aufgaben | 136 |
| 6 Projektumsetzung und Programmierstil | 137 |
| Gute Projektplanung | 137 |
| Codebibliotheken | 145 |
| Frameworks | 146 |
| Code-Wiederverwertung | 146 |
| Guter Programmierstil | 148 |
| Spezialisieren Sie sich | 154 |
| Screendesign – Effektiv kreativ | 156 |
| Aufgaben | 158 |
| 7 Kommunikation | 161 |
| Kommunikations-Overload | 162 |
| Das richtige Medium auswählen | 165 |
| Persönliches Gespräch | 169 |

| | |
|--------------------------------------------------------|------------|
| Verständigung mithilfe von Dokumenten | 169 |
| E-Mail-Kommunikation | 172 |
| Telefonate | 185 |
| Aufgaben | 187 |
| 8 Hard- und Software effektiver einsetzen | 189 |
| Die richtigen Programme | 190 |
| Kompatibilität im Team | 192 |
| Backup | 193 |
| Versionsverwaltung | 194 |
| Stabiler Internetzugang | 195 |
| Hardware optimieren | 197 |
| Aufgaben | 199 |
| 9 Effizienz am Arbeitsplatz | 201 |
| Ordnung erleichtert nicht nur das Finden | 201 |
| Virtueller Schreibtisch | 202 |
| Physischer Schreibtisch | 203 |
| Stifte, Post-its & Co. | 204 |
| Bleiben Sie in Bewegung | 205 |
| Pause vom Monitor | 206 |
| Störfaktoren minimieren | 206 |
| Mobiler Arbeitsplatz | 207 |
| Kleider machen Leute | 208 |
| Aufgaben | 208 |
| Epilog | 209 |
| Belohnen Sie sich | 209 |
| Online-Ressourcen | 210 |
| Index | 219 |

Projektumsetzung und Programmierstil

Mit der Umsetzungsphase eines Projekts beginnt der Teil, in dem man zum ersten Mal Ergebnisse sieht. Wir gehen weg vom Reißbrett und hinein in die Produktionshalle, um die Einzelteile herzustellen und diese zu einem großen Ganzen zusammenzubauen. Viele Webentwickler sind über die Programmierung zu ihrem Beruf gekommen, und somit beginnt hier für viele der Teil, der Spaß macht, bei dem man selbst Herr über die Codezeile ist und bei dem man sich selbst verwirklichen kann. Programmierer verfahren hier nach einem individuellen Muster, das sie über Jahre hin entwickelt haben. Problematisch wird es manchmal mit diesen Mustern, wenn es darum geht, im Team zu arbeiten, fremden Code zu übernehmen und anzupassen oder gar den eigenen Code nach Jahren aus der Vergessenheit hervorzuholen. Rund 80% der Lebenszeit einer Software entfallen auf die Wartung. Bei Webanwendungen ist dieser Wert sehr unterschiedlich je nach Anwendung, die Größenordnung ist allerdings auch hier sicher nicht anders.

Es gibt eine ganze Reihe von Maßnahmen, durch die die Umsetzung vom Screendesign zur Programmierung effektiver ablaufen kann und die anschließende Pflege und Änderung von Code optimiert werden kann. Das geht von der Planung und Strukturierung der Umsetzungsphase über die Wiederverwendung von Elementen und die Einführung eines Workflows beim Screendesign bis hin zum Programmierstil, durch den Sie nicht nur Zeit gewinnen, sondern auch noch Fehler vermeiden. Typische Stichwörter, die in diesem Zusammenhang immer wieder auftauchen, sind Modularisierung, Codebibliotheken, Coding Standards, Frameworks und Code Completion. Damit beschäftigen wir uns in diesem Kapitel.

Gute Projektplanung

In der Phase der Projektumsetzung ernten wir die Früchte, die wir zuvor während der Projektplanung gesät haben. Die Entscheidungen, die wir vorher im Grobkonzept und im Feinkonzept schon getroffen haben, müssen wir jetzt bei der Umsetzung nicht mehr

treffen. Wir können sie in Frage stellen und kritisch bewerten, denn oft ergeben sich aus der Umsetzung Aspekte, die bei der Projektplanung nicht bedacht werden konnten. Obwohl dies eher die Ausnahme sein sollte, ist doch dieses Feedback ein wichtiger Aspekt, der eine Webanwendung erst rund macht. Die Absprachen, die im Vorfeld mit dem Kunden getroffen wurden, und die Analysen über die Zielgruppe sorgen dafür, dass der Code nicht so häufig überarbeitet werden muss und die Entwicklungszeit kurz bleibt.

Entwicklungsdokumentation

Es empfiehlt sich, bereits vor der ersten Zeile Code zu überlegen, wie sich die Dokumentation des Projekts gestalten lässt. Hat man erst einmal ohne Dokumentation angefangen, ist es um einiges schwerer, diese nachträglich in den Workflow einzubetten. Außerdem können konzeptionelle Vorüberlegungen in Form von schematischen Darstellungen schon in die Dokumentation aufgenommen werden. In Kapitel 4, *Projektplanung*, habe ich schon angesprochen, dass das Feinkonzept oft ein lebendiges Dokument ist, das im Laufe des Projekts wächst und quasi eine Entwicklungsdokumentation darstellt. Ab einer gewissen Projektgröße und vor allem, wenn mehrere Entwickler zusammenarbeiten, ist es von Vorteil, eine eigene Entwicklungsdokumentation zu verwenden, um die Aufgabenteilung und -verwaltung zu berücksichtigen. Ich habe schon angesprochen, dass von einer Verlagerung der Entwicklungsdokumentation in den E-Mail-Verkehr abzuraten ist, da dabei nicht gewährleistet ist, dass jeder auf die gleichen Daten zurückgreifen kann. Anbieten würde sich hier eher ein Aufgabensystem oder ein (Bug-)Tracking-System, mit dem einzelnen Entwicklern Aufgaben zugeteilt werden, deren Erledigung die Entwickler gleich im System protokollieren und zur Überprüfung an den zuständigen Teamleiter zurückgeben können.

Als einen Vertreter dafür habe ich das Mantis-System schon in Kapitel 4, *Projektplanung*, kurz beschrieben. Daneben gibt es noch eine große Anzahl anderer Systeme, wie zum Beispiel Bugzilla (www.bugzilla.org), Trac (trac.edgewall.org) oder Track+ (www.trackplus.com).

Wenn Sie für die Programmierung mit einem Versionssystem wie zum Beispiel SVN arbeiten, können Sie die Entwicklungsdokumentation im Tracking-System und die Protokollierung der Änderungen am Code durch die Commit-Kommentare verbinden, indem Sie dort die Tasknummer angeben, durch die im Tracking-System jede Aufgabe eindeutig identifiziert werden kann. Dadurch können Sie sich eine Menge Schreibarbeit sparen, denn Sie müssen nicht den gesamten Aufgabentext auch noch im Subversion-Client eingeben. Die Tasknummer allein reicht aber nicht aus. Mindestens ein paar Stichwörter der Tracking-Aufgabe und der Erfüllungsgrad der Aufgabe sollten zusätzlich noch angegeben werden.

Einen großen Stellenwert nimmt das Protokollieren der Aufgabenlösung in der Entwicklungsdokumentation ein. Dabei kommt es nicht nur darauf an, dass dokumentiert wird, was gemacht wurde. Mindestens ebenso wichtig sind die Begründungen dafür, warum

etwas so gemacht wurde. Nur dadurch ist eine Entwicklung nachzuvollziehen. Einmal getroffene Entscheidungen können neu bewertet werden, und der Fortschritt wird nicht behindert.

Bei kleineren Webprojekten kann die Entwicklung mit Versionen und Meilensteinen in einem HTML-Dokument festgehalten werden, sodass dort dieses HTML-Dokument und das Feinkonzept ausreichend sind. Das Versionssystem kann aus der Softwareentwicklung wie folgt übernommen werden:

```
<major release>.<minor release>.<patch level>-<build number>
```

Die Bedeutung der einzelnen Positionen können Sie Tabelle 6-1 entnehmen.

Tabelle 6-1: Standards bei der Versionsnummerierung

| Versionsnummer | Bedeutung |
|----------------|------------------------------------------------------------------------------------------------------|
| major release | 0 = vor dem Launch ab 1 = nach dem Launch jede weitere Nummer bedeutet ein komplettes Redesign |
| minor release | funktionelle Erweiterung, zum Beispiel durch ein Modul |
| patch level | Fehlerbehebungen |
| build number | einzelner Entwicklungsschritt; ist bei kleinen Projekten nicht unbedingt nötig |

Diese Versionsnummern können in dem HTML-Dokument in Form von Tabellen oder Listen verwendet werden. Jeder Entwickler, sofern es mehrere gibt, dokumentiert mit Namenskürzel und Zeitangabe, was er gemacht hat und wieso er sich für eine bestimmte Technik entschieden hat. Beispiel 6-1 zeigt einen Auszug aus einer HTML-Versionsdatei.

Beispiel 6-1: Auszug aus einer HTML-Versionsdatei

```
<h1>Version 0.2.1</h1>
<ul>
  <li>
    Modul Arbeitszeiten und Monteure gemäß Feinkonzept<br />
    Dateien: daten_eingabe.php, daten_anzeige.php <br />
    Datenbank: Tabellen monteure und arbeitszeiten hinzugefügt <br />
    Begründung: Normalform der Datenbank bleibt erhalten <br />
    TS 081030 22:10
  </li>
</ul>
```

Wie Sie im Beispiel sehen können, wird auf das Feinkonzept verwiesen, und darüber hinaus werden die veränderten Dateien gelistet und die geänderten Datenbanktabellen aufgeführt. Die Änderungen Letzterer werden bei einer Entwicklungsdokumentation häufig übersehen, und nicht selten kommt es dadurch zu auf den ersten Blick nicht nachvollziehbaren Fehlern.

Das HTML-Dokument kann in einem geschützten Bereich auf dem Server allen Projektmitgliedern zur Verfügung gestellt werden. Auch der Kunde kann so den Projektstatus mitverfolgen. Handelt es sich um ein Intranetprojekt, kann das Versionsdokument ein fester Bestandteil werden, der frei zugänglich ist.

Erstellen Sie eine Website mit einer Schnittstelle für den Kunden beziehungsweise Redakteur, damit dieser zum Beispiel bei einem Content-Management-System Inhalte selbst einpflegen kann, sollten Sie unmittelbar während oder nach der Programmierung dieser Schnittstelle auch mit deren Dokumentation beginnen. Die Abläufe sind dort noch präsent und lassen sich leichter dokumentieren. Beim Schreiben der Schnittstellendokumentation fallen eventuell noch vorhandene Fehler oder Optimierungen auf, die sich dann einfacher ausmerzen respektive einfügen lassen, weil Sie noch den richtigen Kontext im Kopf haben. Diese Dokumentation erspart Ihnen viele Rückfragen vonseiten des Kunden. Wenn Sie den Kunden mündlich in die Schnittstelle einweisen, kann dieser oft nicht alles aufnehmen. Außerdem kann es sein, dass nicht alle Bearbeiter der Schnittstelle bei der Einführung anwesend sind.

Eine besondere Rolle hat die Entwicklungsdokumentation bei einem Content-Management-System (CMS). Häufig werden hier Codeblöcke und Templates mit Einstellungsfunktionen der Oberfläche des CMS so vermischt, dass es manchmal viel zu aufwendig wäre, diese Verquickung zu dokumentieren. Vergewöhnen Sie sich noch einmal den Sinn und Zweck von Dokumentationen: Sie sollen Ihnen die Arbeit erleichtern und es einem anderen Entwickler ermöglichen, ohne Probleme später in das Projekt einzusteigen oder es zu übernehmen. Entscheidungen und Entwicklungen sollen zurückverfolgt werden können. Häufig werden diese Probleme durch eine einheitliche Namensgebung von Templates, Navigationshierarchien und Benutzerrechten wesentlich vereinfacht.

Diese Nomenklatur sollten Sie zentral bei dem CMS als Datei speichern, sodass jeder Entwickler, der mit dem Projekt zu tun bekommt, mehr oder weniger automatisch darüber stolpert. Typische Namensgebungen für derartige Informationen sind: *readme*, *Doku*, *snippet*, *et cetera*. Achten Sie allerdings darauf, dass diese Informationen nicht mit veröffentlicht werden. *Doku*-Ordner im Root-Verzeichnis des Live-Servers sind besonders interessant für Angreifer von außen auf die Website.

Modularisierung

Bei der Aufgabenverwaltung in Kapitel 2, *Aufgaben- und Terminverwaltung*, habe ich Ihnen gezeigt, dass sich Aufgaben leichter erledigen lassen, wenn man sie in kleinere Teilaufgaben zerlegt. So ähnlich können Sie es sich hier bei der Umsetzung vorstellen. Wenn Sie die Website mit all ihren Inhalten und Funktionen in Module zerlegen, ist es einfacher, die Einzelteile in Angriff zu nehmen. Neben den einzelnen Seiten können folgende Elemente dabei als Modul gesehen werden: ein PHP-Skript, ein HTML-Template, die Navigation, die Fußzeile, eine PHP-Klasse für den Datenbankzugriff, ein Editor für Textfelder und so weiter.



Sie sehen, dass ich den Begriff »Modul« nicht nur für die objektorientierte Programmierung verwende, in der er eigentlich seinen Ursprung hat, sondern auf alle Inhalte und Funktionen der Site ausweite.

Ein Kritikpunkt der Modularisierung ist, dass die Schnittstellen jedes Moduls, sofern vorhanden, definiert werden müssen. In der Praxis zeigt sich aber fast immer, dass es weniger Aufwand ist, die Schnittstellen festzulegen, als den Überblick über eine einzige große, unübersichtliche Anwendung zu behalten. Spätestens bei der Pflege und beim Ausbau der Site macht sich die Modularisierung bezahlt. Es ist eben einfacher, ein Modul auszutauschen, als die betreffenden Codestellen auf jeder Seite zu verändern.

Eine Kundenverwaltung mit mehreren Ansprechpartnern pro Kunde könnte dann wie in Abbildung 6-1 funktionell in kleinere Module zerlegt werden.

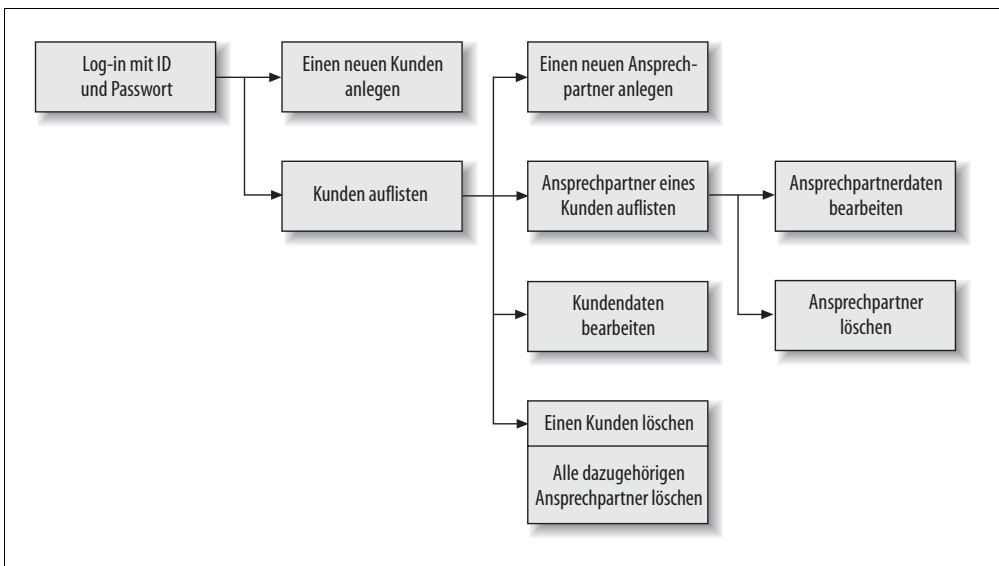


Abbildung 6-1: Funktionelle Modularisierung einer Kundenverwaltung

Aufgrund des so gewonnenen Schemas kann nun überlegt werden, ob Funktionen wie das Neuanlegen, Bearbeiten und Löschen eines Datensatzes in einer Klasse zusammengefasst werden können oder ob das Anzeigen eines Kunden zusammen mit der Auflistung aller Ansprechpartner nicht besser auf einer Seite geschehen soll. Durch die Schema-skizze in diesem sehr einfachen Beispiel lässt sich der Überblick behalten, egal in welche Richtung man weiterentwickelt.

Bei den Modulbeispielen oben habe ich unter anderem Templates aufgezählt. Dies setzt voraus, dass eine Template-Engine verwendet wird. Für manche Sites ist dies zu aufwendig. Sie sollten immer den Aufwand für den Einsatz einer Technik mit der daraus resultierenden Vereinfachung in Relation setzen. Einfacher ist es oft, Teile der Seite mit Funktionen in einer Skriptsprache einzubinden oder gleich mit Includes zu arbeiten. Beispiel 6-2 zeigt, wie das geht.

Beispiel 6-2: Modulaufbau einer Seite mit PHP-Includes

```
<html>
<head>
  <title>Beispiel für Modularisierung</title>
  <link rel="stylesheet" href="standard.css" type="text/css" />
  <style type = "text/css">
    #navi #startseite {background-color: blue;}
  </style>
</head>
<body>
<?php
include("header.php");
include("navi.php");
?>
<!-- Content-Bereich -->
<?php
include("footer.php");
?>
</body>
</html>
```

Header, Navigation und Footer sind in andere Dateien ausgelagert worden. Dort können sie zentral gepflegt, verändert und anschließend in jede HTML-Datei eingebunden werden. Sicherlich gibt es elegantere Methoden. Hier geht es mir nur darum, dass Sie das Prinzip verstehen.

Eine Datenbankstruktur lässt sich meist nicht parallel zu den Modulen in Tabellen aufteilen. Verschiedene Module können durchaus auf dieselben Datentabellen zugreifen. Dies verlangt vor allem bei der Änderung und Erzeugung von Datenbankstrukturen gute Absprachen, wenn unterschiedliche Module von verschiedenen Entwicklern programmiert werden. Die Daten einer Datenbank werden nicht in Module aufgeteilt, sondern in Tabellen, was im weiteren Sinn auch wieder einer Aufteilung in kleinere Bereiche entspricht. Ich kann nur raten, dies konsequent bis zur Normalform der Datenbank zu tun. Dadurch erleben Sie keine bösen Überraschungen, wenn die Datenbank einmal um zusätzliche Daten für neue Funktionen der Website erweitert werden soll. Ein klein wenig mehr Aufwand an dieser Stelle erspart Ihnen später eine große Umstellung bzw. prozessorlastige und umständliche Programme.

Datenbankstrukturen und auch Modularisierung von Funktionen und restlichen Inhalten lassen sich sehr gut in Diagrammen veranschaulichen. Ein Beispiel hierfür sehen Sie in Abbildung 6-2. Durch die Pfeile werden die Relationen zwischen den Tabellen dargestellt. Im Prinzip sollte so ein Diagramm die Ausgangsbasis für die Planung der Module und deren Datenbankabfragen sein. Neben Datenbankstrukturen lassen sich Zusammenhänge von Objekten, Klassen und Methoden aus der objektorientierten Programmierung mittels Diagrammen darstellen. Es gibt Standards für diese Darstellungen. Der bekannteste davon ist die Unified Modeling Language (UML). Da viele gute Bücher zu dieser Sprache auf dem Markt sind, würde es hier zu weit führen, konkret darauf einzugehen. Allerdings dürften sich die wenigsten Kunden oder zum Beispiel ein Grafiker, dem Sie die Anwendungsstruktur erklären müssen, darum kümmern, ob Sie die nach der UML standardisierten Schemata verwenden. Wichtig für diese Personengruppen ist nur, dass der Sachverhalt anschaulich dargestellt ist und schnell erfasst werden kann. Für die Anwendung der UML spricht, dass, wenn die Website sehr skriptlastig ist, UML-Darstellungen den beteiligten Programmierern durchaus helfen können. Wie Sie es machen, bleibt Ihnen überlassen, aber denken Sie daran, eine grafische Darstellung hilft auch Ihnen, den Überblick zu bewahren. Diese Diagramme können Sie wunderbar auch in das Feinkonzept oder die Entwicklungsdokumentation übernehmen. Dadurch ist auch gleichzeitig gewährleistet, dass sie, weil sie nicht übersehen werden können, angepasst werden, sollte sich etwas ändern.

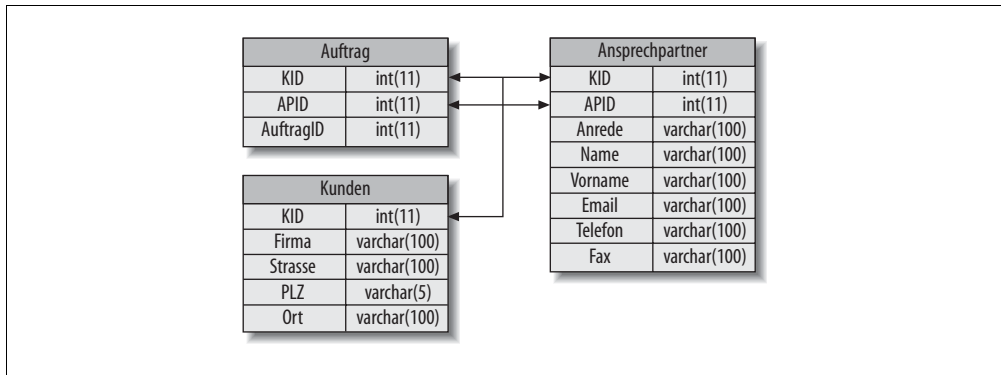


Abbildung 6-2: Beispiel für eine schematische Darstellung einer Datenbankstruktur mit drei Tabellen

Ein weiterer wesentlicher Vorteil der Modularisierung ist, dass dadurch entstandene Teilaufgaben leichter auf mehrere Entwickler verteilt werden können. Somit sind allerdings wieder vermehrt Schnittstellenabsprachen notwendig. Auf der anderen Seite gewinnt die Site an Qualität durch den synergetischen Effekt. Zum Beispiel können schwierige technische Umsetzungen oder aber auch Designs hinsichtlich ihrer Usability und Zielgruppenausrichtung gemeinsam kurz diskutiert werden, um dann wieder von einem Teammitglied

umgesetzt zu werden. Eine andere Methode sieht vor, dass die Entwickler untereinander in Reviews gegenseitig in Teilen oder ganz den Code der anderen durchsehen und anschließend gemeinsam besprechen. Schließlich muss auch noch ein Test der Site oder des Modulabschnitts durchgeführt werden, bevor das Ergebnis dem Kunden präsentiert werden kann. Je enger die Reviewer oder der Tester und der dazugehörige Entwickler zusammenarbeiten, desto besser wird in Zukunft die Qualität des Codes in einer kürzeren Zeit.

In Beispiel 6-2 wurden Codeteile in mehreren Dokumenten wiederverwendet. Genauso ist es auch denkbar, dass Module, wenn sie einen gewissen Abstraktionsgrad erreicht haben, projektübergreifend wiederverwendet werden können. Das wäre beispielsweise bei einer Klasse mit Methoden zum Zugriff auf eine Datenbank möglich. Diese Klasse wird einmal mit einer speziellen Fehlerbehandlung und Überprüfung der Methodenparameter geschrieben, um sie dann in verschiedene Projekte einzubinden und zu verwenden. Die Kunst besteht nun darin, die Klasse und Methoden so allgemein zu halten, dass sie in unterschiedlichen Projekten eingesetzt werden können, sie aber gleichzeitig einen Gewinn beispielsweise gegenüber den standardmäßig von PHP bereitgestellten Datenbankfunktionen darstellen. In Beispiel 6-3 besteht der Gewinn in der Kapselung der Zugangsparameter und der einheitlichen Fehlermeldung.

Beispiel 6-3: Projektübergreifender Modulaufbau

```

class db {
function db() {}
private $dbHost = "localhost";
private $dbUser = "root";
private $dbPass = "password";
private $dbName = "dbname";
private $connect = "";
public function connect($dbHost = '', $dbUser = '', $dbPass = '', $dbName = '') {
    if(!$this->connect = @mysql_connect($this->dbHost, $this->dbUser, $this->dbPass)) {
        die("Keine Verbindung zum Datenbankserver: " . mysql_error());
    }
    if(!$this->selectDB = @mysql_select_db($this->dbName, $this->connect)) {
        die("Datenbank " . $this->dbName . " kann nicht ausgewählt werden: " . mysql_error());
    }
    mysql_query("SET NAMES UTF8");
} // Ende connect
// weitere Funktionen zur Datenbankverwaltung
} // Ende der Klasse db
  
```

Sicherlich gibt es schönere Lösungen, aber hier geht es nur um ein Beispiel. Diese Klasse lässt sich in vielen Anwendungen einsetzen, wenn man die Parameter für die Zugangsdaten jeweils anpasst.

Es gibt noch zahlreiche weitere Anwendungsgebiete für den Einsatz von wiederverwendbarer objektorientierter Programmierung, zum Beispiel Verarbeitung von XML-Dateien, Bereitstellung von Formularfeldern inklusive Überprüfung auf Eingabefehler usw.

Codebibliotheken

Für die wiederverwendbaren Programmteile können Sie sich eine Bibliothek anlegen. Diese kann aus verschiedenen Dateien bestehen, die Sie immer wieder in die einzelnen Projekte kopieren. Kleinere Codefragmente können Sie auch in Snippets im Editor speichern. Sehr schön gelöst ist das zum Beispiel bei Zend Studio für Eclipse (siehe Abbildung 6-3). Die Snippets können in Gruppen zusammengefasst werden. Im Beispiel ist nur die Gruppe *HTML* zu sehen. Neben dem Namen und der Beschreibung können Variablen definiert werden, die im Text im großen unteren Fenster auch mehrfach wiederverwendet werden können.

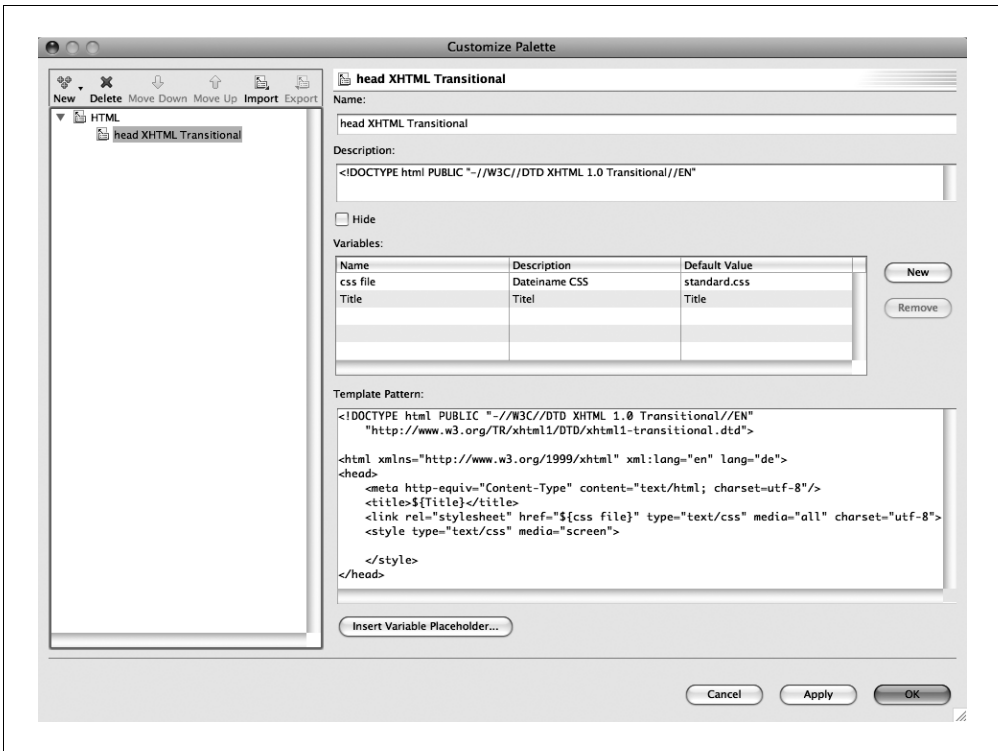


Abbildung 6-3: Snippet-Einstellungsfenster von Zend Studio für Eclipse

Ähnliche, womöglich nicht so umfangreiche Möglichkeiten bieten auch andere Editoren. Wenn Ihr Editor nicht dazugehört, können Sie immer noch die Codefragmente in einer Datei sammeln und sich diese bei Bedarf während des Programmierens per Copy-and-Paste wieder herausziehen.

Sie müssen sich nicht immer die Arbeit machen und Module neu programmieren. Da fast alles schon einmal da war, ist es auch wahrscheinlich, dass das, was Sie brauchen, schon

einmal programmiert wurde. Sie müssen nur wissen, wo Sie danach suchen. Dafür gibt es spezielle Codebibliotheken, wie zum Beispiel die *phpclasses* (www.phpclasses.org). Bitte achten Sie bei der Wiederverwendung von fremdem Code auf folgende Punkte: Halten Sie sich an die Bestimmungen des Urheberrechts. Viele Codebibliotheken sind zwar unter einer Open Source-Lizenz freigegeben, aber vergewissern Sie sich. Es gibt eine heiße Diskussion um das Urheberrecht von Programmcode. Umstritten ist es vor allem, wenn es sich um kleinere Programmteile oder Module handelt. Ich finde, man sollte auf jeden Fall die Wünsche und Lizenzbedingungen des Autors akzeptieren, wenn er den Code schon öffentlich zur Verfügung stellt.

Überprüfen Sie den fremden Code auf Fehler. Manchmal ist die Überprüfung und Anpassung aufwendiger, als würden Sie ihn neu schreiben. Wann dies der Fall ist und wie Sie das im Vorfeld erkennen können, entscheiden Sie aufgrund Ihrer Erfahrung.

Überprüfen Sie unbedingt sicherheitsrelevante Aspekte, wie zum Beispiel die Parameterübergabe oder andere eventuelle Sicherheitslücken. Fehler in weit verbreiteten Modulen aus Codebibliotheken gehören zu den am häufigsten genutzten Angriffspunkten von Hackern.

Frameworks

Im Grunde genommen habe ich in den beiden vorausgegangenen Kapiteln bereits das Prinzip von Frameworks beschrieben; denn in einem Framework werden Objekte und Methoden für bestimmte Anwendungsfälle wie zum Beispiel der Datenbankverarbeitung oder für bestimmte Problembereiche wiederverwendet. Mit der oben gezeigten Datenbankklasse können Sie sich zum Beispiel Ihr eigenes Datenbank-Framework schreiben.

Es gibt aber auch ganz bekannte und sehr ausgereifte Frameworks, wie zum Beispiel das Zend-Framework oder das PEAR-Framework für die Programmierung in PHP oder Prototype, Dojo, Mootools, jQuery und Script.aculo.us für die Programmierung mit JavaScript zum Beispiel für Ajax-Anwendungen.

Der Vorteil bei der Verwendung dieser Frameworks ist, dass man dabei auf einen sehr ausgereiften Code zurückgreift, der je nach Anwendung das Programmieren vereinfachen kann, ohne dass man sich mit den Details der Implementierungen der einzelnen Framework-Komponenten und -Funktionen näher beschäftigen muss.

Code-Wiederverwertung

Nichts liegt näher, als auch auf eigene Codefragmente aus anderen Projekten zurückzugreifen und diese für das aktuelle Projekt anzupassen. Achten Sie hierbei vor allem darauf, dass alle kundenspezifischen Angaben aus dem Code des alten Projekts entfernt werden. Im Groben können Sie das mit der Suchen-und-Ersetzen-Funktion Ihres Editors übernehmen. Überprüfen Sie dennoch den Code noch einmal manuell darauf, ob sich Rechtschreibfehler eingeschlichen haben.

Um das Problem zu umgehen, hat sich Folgendes bewährt: Lagern Sie kundenspezifische Angaben in Konstanten aus, zum Beispiel:

```
define("FIRMENNAME", "TSmedia");
```

Solche Konstanten können Sie in eine zentrale Datei auslagern, die überall per Include eingebunden wird. Diese Lösung bewährt sich auch dann, wenn sich zum Beispiel die Firmierung des Kunden von GmbH in GmbH & Co. KG ändert.

Verwenden Sie eigene Codefragmente mehrmals, sollten Sie daraus einen Eintrag in Ihre Codebibliothek machen oder diese bei den Snippets im Editor aufnehmen.

Ob Sie Codefragmente wiederverwenden oder alles neu programmieren, interessiert den Kunden nicht – daher brauchen Sie es ihm auch nicht zu erzählen. Letztendlich rechnen Sie nach Zeit ab und wollen ein optimales Ergebnis erzielen. Hinzu kommt, dass Sie sich festlegen, wenn Sie dem Kunden eine Technik oder die Verwendung einer Bibliothek vorschlagen. Damit wird es schwieriger, wieder zurückzurudern.

Eines sollten Sie aber auf keinen Fall machen: Ziehen Sie nicht die gesamte Website aus der Schublade. Wenn Sie zu viel von alten Projekten in Ihrem neuen Projekt wiederverwenden, ist es unwahrscheinlich, dass Sie die individuellen Bedürfnisse Ihres Kunden berücksichtigt haben. Ihr Kunde wäre nicht zufrieden, und die Website wäre kein Erfolg, was wieder auf Sie zurückfiele.

Auch die Datenstrukturen von Datenbanken lassen sich zum Teil von anderen Projekten übernehmen. Denken Sie zum Beispiel an ein Adressbuch. Übernehmen Sie aber nur die CREATE TABLE-Anweisung und nicht die tatsächlichen Daten-Inserts. Diese SQL-Befehle können Sie in einer eigenen Datei in der Codebibliothek sichern und immer dann über die Kommandozeile oder Tools wie phpMyAdmin (www.phpmyadmin.net) ausführen lassen, wenn Sie ein Adressbuch benötigen.

Ein kritischer Punkt ist die Weiterentwicklung von eigenen Modulen. Nehmen wir an, dass ein Modul zur Verwendung in ein Projekt A kopiert wird. Bei einem Folgeprojekt B wird das gleiche Modul weiterentwickelt und als verbesserte Kopie eingebunden. Nun möchte der Kunde das Projekt A an das verbesserte Modul angepasst haben, oder Sie selbst möchten diese Anpassung durchführen, um zum Beispiel dadurch ein anderes Modul verwenden zu können, das von dem neuen verbesserten abhängig ist. In diesen Fällen ist es praktisch, wenn Sie in einem Versionsverlauf die Änderungen dokumentiert haben. Interessant sind vor allem geänderte Parameter von Funktionen und Objekten. Auch hier erleichtern ein paar Zeilen Dokumentation das Leben und Programmieren. Projektübergreifende Module können auch derart spezialisiert werden, dass ihre Entwicklungen auseinanderlaufen und sie in unterschiedlichen Projekten eingesetzt werden. Müssen oder sollen nicht alle Verbesserungen des einen Moduls in das andere übernommen werden, sind Komplikationen im Zyklus der Weiterentwicklung des Moduls vorprogrammiert. Entweder wird dann eine detaillierte Entwicklungsdokumentation erforderlich oder die parallele Entwicklung eines zentralen Moduls, in das alle Verbesserungen einfließen.

Guter Programmierstil

Jeder, der schon einmal einen fremden Code überarbeitet hat, kennt das Problem: Unübersichtliche Formatierung, umständliche Variablenbezeichnungen und versteckte Fehler treiben einen zur Verzweiflung. Oft ist das eigentliche Problem, das es zu beheben gilt, trivial, hätte man es erkannt. Dafür war aber der direkte Weg versperrt. Oft formatiert man erst einmal den Code neu und hangelt sich bei der Validierung von Fehler zu Fehler, bis man beim eigentlichen Problem gelandet ist. All das hätte man sich sparen können. Ich würde behaupten, durch einen optimierten Programmierstil lassen sich über 50% der Zeit einsparen gegenüber einem Code ohne System.

Valider Code

Durch das Validieren von Code lassen sich viele Fehler erkennen und im Vorfeld ausschließen. Die Mittel hierfür sind leicht zu implementieren und in den Workflow einzupflegen. Jeder kennt den Validator vom W3C (validator.w3.org) oder validome (www.validome.org). Das System ist einfach: Sie lassen eine Seite, die über das Internet erreichbar ist, durch die Eingabe der Adresse auf Übereinstimmung mit der vorgegebenen oder erkannten Doctype-Definition überprüfen. Ebenso ist es möglich, dass ein HTML-File auf den Server geladen wird, um es dort überprüfen zu lassen. Letzteres ist der einzige Weg, wenn es sich um Seiten aus dem Intranet oder Extranet handelt oder wenn Sie Templates (ohne Platzhalter) validieren möchten. Sind Fehler im Dokument oder auf der Seite, werden diese mit Zeilennummer, Erklärung und weiterführendem Link zur Problemlösung ausgegeben.

Eine andere Möglichkeit für die Validierung sind Programme, die lokal auf Ihrem System vorliegen. Gute Beispiele hierfür sind Dreamweaver, Zend Studio (www.zend.com/de/), aber auch kostenlose Plug-ins für den Firefox-Browser (users.skynet.be/mgueury/mozilla/).

Zend Studio für Eclipse

Die sehr umfassende Validierung von Zend Studio für Eclipse ist modular aufgebaut. Einzelne Module können hinzugeschaltet und installiert werden. Standardmäßig ist schon eine ganze Reihe installiert, zum Beispiel PHP, SQL und XML. Über ein zusätzliches Tidy-Plug-in (eclipsetidy.sourceforge.net/) wird auch HTML-Code bereits während der Eingabe überprüft und gemäß den getroffenen Einstellungen formatiert. Vor allem Letzteres ist ganz praktisch, wenn man Code ohne Zeilenumbrüche übernimmt, da Zend Studio (noch) kein Soft Wrap kann (Version 6.0.1).

Dreamweaver

Das Programm überprüft nicht nur den (X)HTML- und CSS-Code auf Gültigkeit. Es kann sogar eine Browser-Kompatibilitäts-Prüfung durchführen. Treten Fehlermeldungen auf, wird auf eine Wissensdatenbank verwiesen, in der die Problemlösung geschildert wird und diskutiert werden kann. PHP- oder SQL-Code wird leider nicht überprüft. Eine Formatierung des Codes nach eingestellten Regeln ist möglich.

Firefox-Browser-Plug-in

Es gibt mehrere Plug-ins für den Firefox-Browser, die den Code der dargestellten Seite auf Gültigkeit überprüfen. Zu empfehlen ist der HTML Validator (addons.mozilla.org/de/firefox/addon/249), weil Sie zwischen zwei Algorithmen wählen oder beide nacheinander ausführen können: HTML tidy und SGML Parser. Die Überprüfung findet lokal auf Ihrem Computer statt. Treten Fehler auf, werden diese mit der Zeilennummer, einer Referenz und einem Lösungsansatz ausgegeben. Ebenfalls möglich ist es, den Quelltext zu formatieren.

Somit haben Sie immer die Möglichkeit, Ihren Code zu überprüfen. Und das kann gar nicht oft genug sein. Durch validen Code umgehen Sie viele Browser- und Darstellungsprobleme. Besonders wichtig ist die Validierung von Templates (ohne die Platzhalter), die Sie zum Beispiel in Content-Management-Systemen benutzen. Versteckt sich schon im Template ein Fehler, finden Sie nach dem Einfügen von Containern und sonstigem Code durch das CMS den Fehler um so schwerer. Am einfachsten ist es, wenn Sie das Template vor dem Einfügen in das CMS überprüfen.

Eine eigene Anwendung zur Formatierung von Quellcode können Sie sich mit dem PHP-Plug-in Tidy (de2.php.net/manual/de/book.tidy.php) zulegen. Damit werden zahlreiche Funktionen zur Verfügung gestellt, mit denen HTML-Quelltext formatiert werden kann.

Coding Standards

Selbstständige Programmierer sind vielleicht nur dann einmal mit Coding Standards in Kontakt gekommen, wenn sie gemeinsam mit anderen Entwicklern Code zu einem größeren Projekt beigetragen haben. Eventuell empfanden sie diese Vorschrift dann eher als lästig denn als nützlich. Sobald aber mehrere Programmierer an einem Projekt arbeiten und speziell wenn der Code von anderen gewartet werden soll, sind Coding Standards sehr nützlich. Aber auch der Einzelkämpfer tut sich einen Gefallen, wenn er für sich selbst einen Coding Standard entwirft oder seinen Programmierstil an den von Kollegen anpasst, mit denen er eventuell einmal zusammen arbeiten könnte. Copy-and-Paste-Aktionen oder projektübergreifendes Wiederverwenden von Code beansprucht viel weniger Zeit, weil der Code nicht neu formatiert werden muss oder Variablen teilweise nicht umbenannt werden müssen.

In diesen Vorschriften sollten Sie zum Beispiel festlegen, dass Quelltexte grundsätzlich mit vierstelligen Tabs eingerückt werden und eine maximale Zeichenzahl von 80 Zeichen pro Zeile haben dürfen. Andere Alternativen wären: Anstatt Tabs werden Leerzeichen verwendet, wie sie von einigen Editoren automatisch eingefügt werden, oder die Zeilen werden gar nicht manuell umbrochen, sondern vom Editor automatisch per Soft Wrap umbrochen dargestellt. Letztere Möglichkeit ist problematisch, wenn Entwickler andere Editoren verwenden wie zum Beispiel Zend Studio, die diese Funktion nicht besitzen.

Durch diese Formatierungen lassen sich Codezusammengehörigkeiten viel einfacher überprüfen, und fehlende schließende Tags können leichter erkannt werden. Hierbei helfen vor allem bei sehr weit auseinanderstehenden Start- und End-Tags zusätzliche Kommentare, die nach dem schließenden Tag im Kommentartext die `id` des Start-Tags wiederholen:

```
<div id="name">
... ganz viel Code ...
</div> <!-- end name -->
```

Dies kann man nicht nur mit `div`-Tags und nicht nur im HTML-Code machen, sondern auch mit Klammern von Schleifen oder `if`-Anweisungen im PHP-Code. Dadurch werden schließende Tags oder Klammern auch nicht so schnell aus Versehen gelöscht. Um Code auszuwählen und ihn zu kopieren, sind Kommentare ebenfalls sehr hilfreich. Eventuell lassen sich diese Kommentare schon bei der Code Completion einstellen, aber dazu später mehr in diesem Kapitel.

Das Format und der Inhalt von Kommentaren im Quelltext sind nicht nur durch Syntaxdefinitionen geregelt. Verwendet man innerhalb der Kommentare einen einheitlichen Standard, können die Inhalte zum Beispiel für die automatische Erstellung einer Dokumentation verwendet werden. Ein System dafür ist der `phpDocumentor` (www.phpdoc.org). Diese PHP-Anwendung erstellt aufgrund von Tags in der Schreibweise `@tagname`, die sich in den Kommentaren befinden, Dokumente in den Formaten HTML, PDF, CHM oder XML DocBook. Am Anfang von Dateien und vor jeder Funktion, Klasse und Variablendefinition sind speziell formatierte Kommentare einzufügen, wie Beispiel 6-4 zeigt:

Beispiel 6-4: Kommentare im phpDocumentor

```
<?php
/**
 * Beschreibung auf DocBlock-Ebene
 *
 * @package foo functions
 * @author Thomas Steglich <email@tsmedia.de>
 * @version 1.13
 */

/**
 * prüft, ob string gleich foo ist
 *
 * @param string eingabe
 * @return bool
 */
function check($eingabe){
    return($eingabe == "foo");
}
?>
```

Mittels eines Kommandozeilenbefehls oder einer PHP-Funktion kann nun je nach Parameter automatisch eine Dokumentation erzeugt werden. Bevor Sie aber ein ganzes Projekt mit diesen Kommentaren dokumentieren, sollten Sie die Ausgabe mit einem Beispiel testen. So können Sie sich auch besser entscheiden, in welcher Form Sie sich die Dokumentation ausgeben lassen. Die HTML-Variante lässt sich leichter auf dem Testserver parallel zum Projekt bereitstellen, die PDF-Version sieht ausgedruckt schöner aus, die CHM-Datei lässt sich eventuell als Hilfe installieren, und die DocBook-Dokumentation lässt sich leichter ergänzen, wenn man sich mit diesem Dokumentformat auskennt und einen geeigneten XML-Editor besitzt. Zend Studio hat die phpDocumentor-Funktion eingebaut. Ihre gesamte Dokumentation können Sie so in den Quelltext schreiben, um anschließend nur aufs Knöpfchen zu drücken und die Dokumentation für den Kunden in den Händen zu halten. In der Praxis muss diese Dokumentation dennoch um einige Angaben ergänzt werden. Auf jeden Fall nimmt einem diese Methode viel Arbeit ab.

Die Kommentare oben befinden sich ausschließlich im PHP-Code. Vermeiden Sie Entwicklerkommentare im HTML-Codeteil. Diese wären für den Benutzer, der auch einen Blick in den Quellcode wirft, sichtbar und würden so ein Sicherheitsrisiko darstellen.

In einem Coding Standard sollten die Schreibweisen von Namen für Variablen, Klassen, Methoden und Konstanten definiert sein. Warum sollten Sie so etwas tun? PHP unterscheidet zum Beispiel bei Bezeichnern zwischen Groß- und Kleinschreibung. Es gibt aber unterschiedliche Methoden, einer Variablen einen aussagekräftigen Namen zu geben:

- UpperCamelCase-Notation, Beispiel: GetValueOfAttribut
- lowerCamelCase- oder camelCase-Notation, Beispiel: getValueOfAttribut
- mit Unterstrich und alles kleingeschrieben: get_value_of_attribut

Wenn Sie diese Möglichkeiten im Coding Standard festlegen, müssen Sie nicht immer überlegen, wie Sie den jeweiligen Namen vergeben haben, wenn Sie ihn in einer anderen Datei verwenden. Die Sprache, die bei den Namen verwendet wird, kann auch festgelegt werden. Handelt es sich um ein Projekt, an dem mehrere Nationalitäten beteiligt sind, würde sich Englisch als Standardsprache anbieten. Sind aber nur deutsche Entwickler beteiligt, kann für diese Nomenklatur auch ausschließlich die deutsche Sprache verwendet werden. Damit sich Konstanten eindeutig von Variablen abheben, sollten Sie diese einheitlich großschreiben, zum Beispiel KONSTANTE.

Sich innerhalb eines Projekts auf einen Doctype und eine Kodierung festzulegen, ergibt durchaus Sinn. Dadurch sind Codeteile in mehreren Dateien sicher universell verwendbar. Wenn Sie nun aber mit Codebibliotheken projektübergreifend arbeiten, ist es von Vorteil, wenn Sie sich in den Coding Standards auf einen Doctype einigen. Vor allem zwischen HTML und XHTML lassen sich nicht so ohne weiteres Codeteile austauschen. Einen gewissen Vorteil haben Sie bereits, wenn Sie auch bei HTML-Doctype-Dateien die Tags kleinschreiben. Grundsätzlich sollte Ihr Doctype auf dem aktuellen Stand der Technik

sein, d.h., Sie sollten normalerweise besser mit XHTML als mit HTML arbeiten. Verwenden Sie aber Bibliotheken, die auf einem anderen Standard aufgebaut sind, kann es unter Umständen erforderlich sein, diesen für das ganze Projekt zu übernehmen. Es sollte also auch für Coding Standards Ausnahmen geben.

Einige Entwickler plädieren dafür, bei den Attributen in CSS-Stildefinitionen eine Reihenfolge vorzugeben. Dabei hat sich die alphabetische Reihenfolge gegenüber anderen Methoden behaupten können. Dagegen spricht wiederum, dass sich dann die Positionsangaben mit den Formatierungsanweisungen zu sehr mischen. Bei der Reihenfolge der Stile sollten Sie auf jeden Fall ein System haben. Dies ist sinnvoll, weil Stylesheets mitunter sehr lang werden können. Um hier eine übersichtliche Struktur zu bekommen, empfehle ich, die Attribute einzurücken und die Elemente, Klassen und IDs jeweils für sich zusammenzufassen. Trennen Sie einzelne Blöcke durch Kommentare und schreiben Sie in die Kommentare den Verwendungszweck der Stildefinitionen. Praktisch ist es auch, die Farben, die sich aus dem Corporate Design ergeben und die im Projekt immer wieder auftreten, am Anfang des Standard-Stylesheets in einem Kommentarblock festzuhalten. Diese Stelle dient dann immer wieder dem Nachschlagen und ist praktisch für Copy-and-Paste. All dies kann in einem Coding Standard festgelegt werden. Dort sollte dann auch stehen, dass die Styles möglichst zentral in Stylesheets zusammengefasst werden anstatt als Inline-Stildefinition. Eine solche CSS-Datei könnte gegliedert sein wie in Beispiel 6-5 gezeigt:

Beispiel 6-5: Eine standardorientierte CSS-Datei

```
/* Farbschemata
Seitenhintergrund: #D5D5D5
Seitenrahmen: #a8a8a8
Menü Hintergrund: #A1E580
Link hover und Überschriften: #5351FF
Metabox Hintergrund: #E4E4E4
*/

* {
    border: 0;
    margin: 0;
    padding: 0;
}

body {
    background-color: #D5D5D5;
    font-size: 75%;
}

html>body {
    font-size: 12px;
}

a:link {
    color: black;
}
```

Beispiel 6-5: Eine standardorientierte CSS-Datei (Fortsetzung)

```
a:visited {
    color: #727272;
}
a:hover, #menu a:hover {
    color: #5351FF;
}
a:active {
    color: red;
}
a:focus {
    outline: none;
}
p {
    font: 1em/1.5em Arial, Helvetica, sans-serif;
    margin-bottom: 1.5em;
}
/* weitere Stildefinitionen für Tags */
.meldung {
    color: red;
    font-weight: bold;
}
/* weitere Stildefinitionen für Klassen */
#wrapper {
    background-color: white;
    border: 1px outset #a8a8a8;
    margin: 10px auto;
    padding: 10px;
    width: 780px;
}
/* weitere Stildefinitionen für Individualformate */
```

Einige Beispiele für verbreitete Coding Standards werden im Folgenden kurz gelistet mit jeweils ein paar Stichpunkten zu den wesentlichen Vereinbarungen:

- Zend Framework Coding Standard for PHP (framework.zend.com/manual/en/coding-standard.html): Einrückung mit vier Leerzeichen, keine Tabulatoren; Zeilenlänge von 80 Zeichen; Variablen in der camelCase-Notation; Konstanten durchgehend großgeschrieben; phpDocumentor-Standard
- PEAR Coding Standard (pear.php.net/manual/de/standards.php): Einrückung mit vier Leerzeichen, keine Tabulatoren; Zeilenlänge von 80 Zeichen; phpDocumentor; Variablen und Funktionsnamen in camelCase-Notation
- Coding Standard von Claus van Beek (php-coding-standard.de): Kommentare und Bezeichner in englischer Sprache; Änderungskommentare mit Zeitstempel in jeder Datei; jede Funktion mit Kommentar; Benennung von Variablen und Funktionen in Kleinbuchstaben mit Unterstrich; Konstanten in Großbuchstaben; Parameter und Rückgabewerte von Funktionen werden dokumentiert; Templates müssen validiert werden; Einrückung mit Tabulator

- Coding Standard von Fredrik Kristiansen (www.dagbladet.no/development/phpcodingstandard/): Funktionsbezeichner in der CamelCase-Notation; Variablenamen mit Unterstrich und Kleinschreibung; Einrückung mit vier Leerzeichen; jedes Verzeichnis hat eine erklärende readme-Datei

Syntax-Highlighting

Diese Funktion ist Standard in fast allen Quelltexteditoren und erleichtert die Suche von Syntaxfehlern ungemein. Bei einigen Browsern können die einzelnen Farben vom Benutzer definiert werden. Interessant ist dies vor allem dann, wenn Sie den Hintergrund auf eine dunkle Farbe einstellen und die Schrift hell oder auch weiß einstellen. Der insgesamt dunkle Eindruck beruhigt die Augen und lässt diese nicht so schnell müde werden. Probieren Sie es doch einmal aus. Zu grelle Farben sollten Sie deshalb ebenfalls vermeiden.

Benutzer von älteren Browsern freuen sich, wenn sie zu neueren wechseln, in denen auch die Quelltextansicht ein Syntax-Highlighting besitzt, wie zum Beispiel Firefox 3 oder IE 8. In Internet Explorern früherer Versionen lässt sich diese mittels des Programms Notepad2 (www.flos-freeware.ch/notepad2.html) als Plug-in installieren.

Code Completion

Die Funktion, die beim Schreiben von Code Tags und eventuell sogar PHP-Funktionen vorgeschlägt, und zwar entweder per automatischem Pop-up oder durch eine Tastenkombination, haben schon einige Editoren implementiert, so zum Beispiel Dreamweaver und Textmate/E-Texteditor. Dass aber auch Variablen und Benutzerfunktionen im aktuellen Sichtbarkeitsbereich des jeweiligen Projekts angeboten werden, gibt es bisher nur bei wenigen Editoren. Letztere Funktion finden Sie zum Beispiel bei Zend Studio for Eclipse. Sie erleichtert nicht nur die Codeeingabe, sie verhindert auch die falsche Schreibweise bei Variablen und Funktionsbezeichnern. Diese Fehler sind häufig verantwortlich für eine zeitaufwendige Suche. Somit bieten diese Editormerkmale auch echten Zeitgewinn.

Spezialisieren Sie sich

Sie können nicht in allen Spezialgebieten der Webentwicklung alles wissen. Diese These oder Tatsache habe ich schon mehrmals in diesem Buch proklamiert. Es ist wichtig zu wissen, was Sie mit welcher Technik anstellen können und welche Voraussetzungen dafür notwendig sind. Um aber bei der Umsetzung wenigstens mitreden zu können, sollten Sie in jedem Bereich mindestens eine Technik inklusive Realisierung beherrschen. Es spricht nichts dagegen, wenn Sie mehrere Techniken pro Sparte anwenden können. Bedenken Sie aber, dass Sie bei einem Wechsel der Technik von einem Projekt zum nächsten eine gewisse Einarbeitungszeit berücksichtigen müssen. Diese können Sie vermeiden,

wenn Sie eine Technologie favorisieren und die anderen umsetzen, indem Sie die Teilaufgaben an Spezialisten delegieren. Bei der Auswahl einer Technologie wird zwischen folgenden Bereichen unterschieden: (X)HTML und CSS, server- und clientseitige Programmierung von Webanwendungen, Datenbankanwendung und Layout. Sie stehen damit vor der Entscheidung, sich in jedem Bereich mindestens eine Technik auszusuchen, bei der Sie tiefer in die Materie einsteigen. Im Folgenden möchte ich Ihnen eine kurze Hilfestellung für diese Entscheidungen geben.

Das Votum für HTML oder XHTML ist, wie ich weiter oben schon angedeutet habe, unter anderem abhängig von fremdem Code, auf den Sie bei der Umsetzung des Projekts angewiesen sind. Sind Sie aber frei in der Entscheidung, müssen Sie die eine Technologie gegenüber der anderen abwägen.

Die Vorteile von HTML sind, dass Sie vielleicht schon jahrelang diesen Typ verwenden und damit bestens vertraut sind. XHTML wird vom W3C (www.w3.org) seit Januar 2000 empfohlen, da es mit anderen wichtigen XML-Sprachen kompatibel ist. Der Stellenwert von XHTML wird in der Zukunft steigen und der Anteil an XHTML-Seiten wachsen. Für Sie stellt sich die Frage, wann Sie auf diesen Zug aufspringen, wenn Sie es nicht schon getan haben, und was Sie dabei beachten müssen.



Eine gute Übersicht über das Für und Wider bietet die Website von SelfHTML (de.selfhtml.org/html/xhtml/unterschiede.htm).

Als echte Skriptsprachen, die auf der Clientseite ausgeführt werden, stehen JavaScript, VBScript und ActionScript zur Auswahl. Clientseitiges Visual Basic Script (VBScript) kann nur auf Microsoft-Clients ausgeführt werden, ist dann aber ein mächtiges Werkzeug, das viele bereits installierte Bibliotheken nutzen kann. ActionScript ist mit Adobe verheiratet und benötigt Flash, Flex oder Air, um ausgeführt zu werden. Entsprechende Plug-ins gibt es für die meisten Browser. Aufgrund seiner Verbreitung und seiner standardmäßigen Implementierung in fast allen Browsern kommt JavaScript aber die bedeutendste Rolle zu. Noch mehr Gewicht bekommt es durch die steigende Verbreitung von Ajax (Asynchronous JavaScript and XML). Aufgrund der technologisch unterschiedlichen Bedeutung sollten Sie grundsätzlich wissen, für welche Anwendung Sie welche Skriptsprache einsetzen können. Wenn es aber darum geht, dass Sie sich dafür entscheiden müssen, in welche Technologie Sie sich tiefer einarbeiten wollen, um diese ständig parat zu haben, und wenn Sie sich nicht zu sehr spezialisieren möchten, hat also JavaScript die Nase vorn. Durch die Entscheidung für eine Skriptsprache, und damit die speziellen Anwendungen den Spezialisten zu überlassen, haben Sie den Vorteil, sich auf eine Technik zu konzentrieren, und die Gefahr einer Verwechslung von Merkmalen ist nicht so groß.

Auf der Serverseite ist die Auswahl an Skriptsprachen schon größer, und die Marktanteile sind hier nicht so eindeutig. Daher ist die Entscheidung für nur eine Sprache weder einfach noch wirklich ratsam. Wichtig ist, dass Sie sich auch hier der Stärken und Schwächen der einzelnen Sprachen im Vorfeld von Projekten bereits bewusst sind. In mindestens einer dieser Sprachen sollten Sie so weit eingearbeitet sein, dass Sie Anwendungen selbst erstellen oder fremden Code hinsichtlich seiner Funktion beurteilen können: PHP, Ruby mit Framework Ruby on Rails, Perl, Python, SSI, Cold Fusion usw. Der Anteil der PHP-Anwendungen dürfte zurzeit der größte sein, aber Sie können ebenso gut Pluspunkte bei der Vergabe von Aufträgen sammeln, wenn Sie auf eine andere Sprache spezialisiert sind, weil es eben dann von diesen Spezialisten nicht so viele gibt.

Im folgenden Abschnitt soll es um die Auswahl der Technik beim Screendesign gehen. Auch in diesem Bereich gibt es Optimierungsmöglichkeiten, die Ihnen dabei helfen, Ihre Zeit effektiv zu nutzen.

Screendesign – Effektiv kreativ

Genau so wie es bei dem ganzen Projekt einen bestimmten Workflow gibt, der eine effektive Projektentwicklung möglich macht, gibt es auch beim Screendesign Abläufe, deren richtige Reihenfolge doppelte Arbeiten vermeiden hilft, denn dadurch können die einzelnen Teilaufgaben wirkungsvoller umgesetzt werden. Im Folgenden gehe ich Schritt für Schritt auf die einzelnen Phasen ein.

Ausgangsbasis für ein Screendesign ist das Corporate Design (CD). Ist dies für ein Projekt noch nicht festgelegt, besteht die erste Aufgabe darin, es zu entwickeln. Dabei müssen Logo, Schrift und Farben festgelegt werden. Dies kann ein langwieriger Prozess sein, bei dem viele Absprachen und mehrere Präsentationen notwendig sind. Eine Beschleunigung dieses Prozesses ist schwer durchzusetzen, da Sie hier stark vom Kunden und dessen Vorstellungen abhängig sind. Für Ihre Projektkalkulation ist wichtig, dass Sie sich die hierfür notwendige Zeit einplanen und einen großen Zeitpuffer berücksichtigen.

Stehen die Farben des CD fest, können Sie auf der Basis der Zielgruppenanalyse zusätzliche Farben festlegen, die zu den CD-Farben harmonisch passen. Legen Sie sich aber auf Farben für die Website fest, bevor die CD-Farben definiert sind, engen Sie sich entweder in der Farbgebung bei den CD-Farben ein, oder Sie müssen die Farben für die Website nachbessern. In beiden Fällen haben Sie dann mit einem Mehraufwand zu rechnen. Werkzeuge für die harmonische Farbabstimmung finden Sie bei Illustrator im Menü *Farbhilfe* beziehungsweise *Interaktive Farbe* oder bei Photoshop ab Version CS4 Extended. Adobe bietet auf einer Website mit dem Dienst Kuler (kuler.adobe.com) eine ähnliche Funktion an. Dort können Benutzer zusätzlich eigene Favoriten hochladen und anderen Benutzern zur Verfügung stellen. Diese Bibliothek soll Sie allerdings nur bei der eigenen Komposition unterstützen, denn es ist unwahrscheinlich, dass bei einer bereitgestellten Farbharmonie bereits die CD-Farben Ihres Projekts berücksichtigt wurden. In seltenen Fällen ist die Basis für

eine Farbharmone ein Foto. Dann helfen Ihnen die bereits genannten Möglichkeiten nur bedingt weiter. Allerdings gibt es auch für diese Situation ein Programm: Die Mac OS X-Software Color Schemer Studio (www.colorschemer.com) erstellt auf der Grundlage eines Fotos Farbharmone, die Sie dann für Ihr Layout übernehmen können. Es kann natürlich sein, dass Sie ab einer gewissen Stufe im Screendesign-Workflow wieder hierher zurückkommen müssen, um das Farbschema zu korrigieren, weil die gewünschte Wirkung im (halb-)fertigen Layout verfehlt wird. Um aber mit dem Layout beginnen zu können, benötigen Sie eine Auswahl an Farben, die Sie verwenden können. Wie schon zuvor in diesem Kapitel beschrieben, tun Sie sich einen Gefallen, wenn Sie die entsprechenden Farben als RGB- oder Hex-Werte am Anfang des Standard-Stylesheets in einem Kommentar notieren.

Die grobe Aufteilung der Funktionen und Inhalte einer Website können Sie mit einem Skribble vornehmen. In einer freihändig gezeichneten Skizze verteilen Sie Logo, Header, Navigationsleisten und typische Inhalte auf der Webseite. Die Meinungen gehen auseinander, ob dieses Skribble dazu geeignet ist, dem Kunden bereits in diesem Stadium einen Eindruck von der fertigen Website zu vermitteln. Der Vorteil des Skibbles ist jedoch, dass es schnell angefertigt ist, einfach durch eine neue Version revidiert werden kann und weniger endgültig wirkt. Nachteilig ist, dass es dem Kunden mehr Vorstellungskraft abverlangt, die manchmal nicht vorhanden ist. Je nach Qualität der Zeichnung bildet sich der Kunde möglicherweise seine eigene Meinung bezüglich Ihrer professionellen Herangehensweise. Es kann sein, dass der Kunde bereits daran gewöhnt ist, fertige Layouts präsentiert zu bekommen. Gleichgültig, ob Sie das Skribble dem Kunden präsentieren oder nicht, für Sie ist es eine hilfreiche Unterstützung, das Layout weiterzuentwickeln.

Genau so wie es bei den Farben auf Harmonien ankommt, gibt es auch bei der Rastereinteilung des Layouts Harmonien, die Sie berücksichtigen sollten, um dem Benutzer ein stimmiges Gesamtbild liefern zu können. Die Bestimmung des Rasters ist der nächste Schritt. Hilfestellung bekommen Sie hier durch die »Methode der grauen Boxen«, die ich Ihnen schon in Kapitel 6, *Projektumsetzung und Programmierstil*, vorgestellt habe. Aufgabe ist es nun, die Boxen hinsichtlich ihrer Größe harmonisch in ein Raster einzufügen. Die bekannteste Einteilung ist der Goldene Schnitt: Zwei Strecken stehen im Verhältnis des Goldenen Schnitts zueinander, wenn sich die größere zur kleineren verhält wie die Summe der beiden zur größeren.

Mit dem Werkzeug Ihrer Wahl können Sie oder ein Grafiker nun das Layout anfertigen. Im Wesentlichen stehen hier die Programme Photoshop, Illustrator und Fireworks zur Verfügung. Alle drei werden von der Firma Adobe (www.adobe.com/de) hergestellt. Grundsätzlich ist ein Layout mit jedem dieser drei Vertreter möglich. Beachten sollten Sie bei der Auswahl des Tools jedoch, dass zum Beispiel Photoshop eher pixelorientiert ist, wohingegen die anderen beiden ihre Stärken bei Vektorgrafiken ausspielen können. Je nachdem, wie groß der Anteil der Fotos an Ihrem Layoutkonzept ist und ob Sie spezielle Bitmap-Effekte einbauen, wählen Sie zwischen den Alternativen. Denken Sie (oder der

Grafiker) bereits bei der Umsetzung im Grafikprogramm daran, dass Sie das Layout in Bereiche für den Übergang zum CSS-gesteuerten HTML-Code einteilen. Dies geht am einfachsten mit Slices und Ebenen beziehungsweise Ebenenkompositionen (Photoshop). Die Slices ermöglichen es Ihnen, bei nachträglichen Änderungen die einzelnen Bereiche wieder in der richtigen Größe und an der vorher definierten Position auszuschneiden und in die CSS-Komposition einzufügen.

Den Übergang vom Layout zur CSS- und HTML-Programmierung sollten Sie anhand von Standardvorlagen aus Ihrer Codebibliothek trainieren. Wenn Sie diese Transformation schon ein paar Mal gemacht haben, werden Sie Parallelen erkennen, und die Arbeit geht Ihnen schneller von der Hand. Auch hilft Ihnen hier die Erfahrung bei der Auswahl der Vorlagen und Techniken und der anschließenden Anpassung an das Layout. Berücksichtigen Sie bereits bei der Vorlagenauswahl die Layoutkombinationen für andere Seiten. Dadurch reduzieren Sie die Anzahl der Stylesheet-Anweisungen für die Positionierung und sind flexibler für zukünftige Seiten, durch die die Website ergänzt werden könnte.

Aufgaben

1. Sehen Sie sich die Demos der Tracking-Tools an. Notieren Sie sich dabei die Vor- und Nachteile, die sich für Ihre spezielle Situation ergeben. Füllen Sie daraufhin die Entscheidung, ob und, wenn ja, welches der Werkzeuge Sie in Zukunft einsetzen wollen. Richten Sie gegebenenfalls das ausgewählte Werkzeug auf Ihrem Server ein.
2. Üben Sie bei Ihrem nächsten Projekt die Aufteilung in Module. Erstellen Sie dazu aussagekräftige schematische Darstellungen, die Sie auch Ihrem Kunden präsentieren können. Übernehmen Sie die Schemata in Ihr Feinkonzept.
3. Legen Sie eine Codebibliothek an und füllen Sie diese nach und nach mit Vorlagen, die Sie auch wirklich verwenden. Integrieren Sie Snippets in den Editor Ihrer Wahl.
4. Besuchen Sie Open Source-Codebibliotheken im Internet. Überprüfen Sie die dort bereitgestellten Möglichkeiten und ob der jeweilige Code hinsichtlich Funktion und Qualität Ihren Ansprüchen genügt. Nehmen Sie gegebenenfalls die Bibliothek in Ihre Bookmarks auf.
5. Überprüfen Sie die Möglichkeiten für die Validierung von Code, die sich Ihnen in Ihrem Workflow und/oder Ihrem Editor bieten. Verwenden Sie die Gültigkeitsüberprüfung so oft wie möglich, damit sie fester Bestandteil Ihres Workflows wird.
6. Erstellen Sie für sich oder Ihr Team einen Coding Standard. Stellen Sie das Dokument allen Teammitgliedern über eine zentrale Stelle, zum Beispiel Intranet oder Extranet, zur Verfügung. Machen Sie Ihr Team auf Aktualisierungen per E-Mail aufmerksam.

7. Überprüfen Sie die Einstellungen an Ihrem Editor für Syntax-Highlighting und Code Completion. Sollte Ihr Editor die jeweiligen Funktionen nicht bieten, sehen Sie sich diese in Demoversionen von anderen Editoren an und wägen einen Wechsel ab, um in Zukunft schneller fehlerfreien Code entwickeln zu können.
8. Überprüfen Sie Ihr Fachwissen bezüglich Skriptsprachen und füllen Sie Lücken, indem Sie die entsprechenden Weiterbildungsmaßnahmen in Ihre Aufgabenverwaltung aufnehmen.
9. Entwickeln Sie für sich oder Ihr Team einen festen Workflow für das Screendesign und wenden Sie ihn beim nächsten Projekt gleich an.