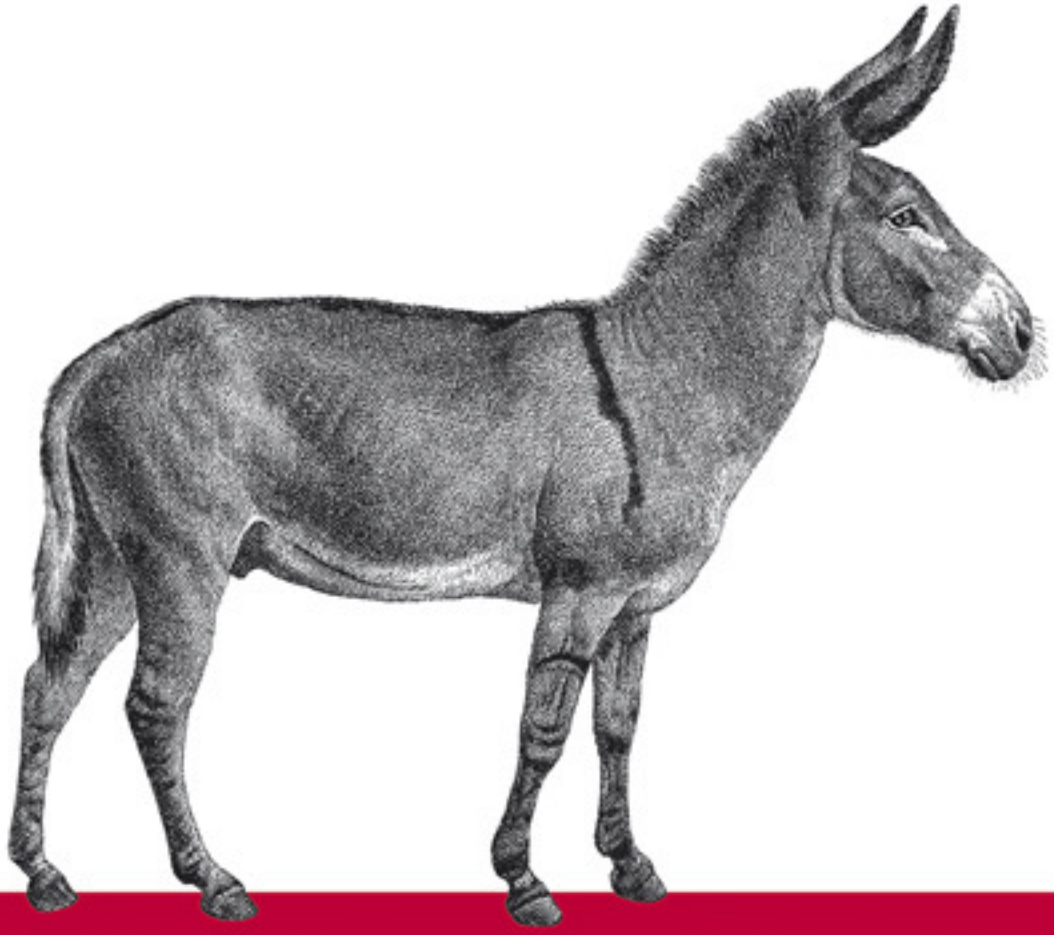
O'REILLY®



Weniger schlecht programmieren

Kathrin Passig &
Johannes Jander

Kathrin Passig / Johannes Jander, Weniger schlecht programmieren, O´Reilly, ISBN 9783897215672978389721567297

Inhalt

Teil 4: Wahl der Mittel

KAPITEL 19

Mach es nicht selbst

> »In Polen lebte einmal ein armer Jude, der hatte kein Geld, zu studieren, aber die Mathematik brannte ihm im Gehirn. Er las, was er bekommen konnte, die paar spärlichen Bücher, und er studierte und dachte, dachte für sich weiter. Und erfand eines Tages etwas, er entdeckte es, ein ganz neues System, und er fühlte: ich habe etwas gefunden. Und als er seine kleine Stadt verließ und in die Welt hinauskam, da sah er neue Bücher, und das, was er für sich entdeckt hatte, das gab es bereits: es war die Differentialrechnung. Und da starb er. Die Leute sagen: an der Schwindsucht. Aber er ist nicht an der Schwindsucht gestorben.«
>
> Kurt Tucholsky, »Es gibt keinen Neuschnee«

Unerfahrene Programmierer bringen viel Zeit damit zu, Funktionen neu zu erfinden, die es in ihrer Sprache oder deren Standardbibliotheken bereits gibt. Natürlich ist es kaum möglich, von Anfang an einen Überblick über alle Funktionen zu haben, die die Programmiersprache mitbringt. Niemand liest sich als Erstes die alphabetische Auflistung sämtlicher Befehle einer Sprache.[1] Wie können Sie trotzdem ohne großen Aufwand feststellen, wo sich eigene Nachdenk- und Programmierarbeit lohnt und wo Sie nur das Rad neu erfinden (in Form von aneinandergenagelten Dreiecken)?

> »Unvollständige Liste der PHP-Befehle, die ich aus Unwissenheit selbst nachgebaut habe: `array_rand`, `disk_free_space`, `file_get_contents`, `file_put_contents`, `filter_var`, `htmlspecialchars`, `import_request_variables`, `localeconv`, `number_format`, `parse_url`, `strip_tags`, `wordwrap`.«
>
> Kathrin

Es sind nicht nur Unerfahrenheit und Unwissenheit, die nicht so gute Programmierer von der Verwendung bestehender Lösungen abhalten. Selbst wenn eine Aufgabe im Programmierer den vagen Verdacht erweckt, dieses Problem könnte eventuell schon einmal ein anderer Mensch gehabt und gelöst haben, folgt daraus nicht unbedingt der naheliegende Schritt, nach der Lösung zu suchen. Viele Menschen empfinden Programmieren als die

1 Fast niemand. Aber eigentlich ist es sogar eine ganz gute Idee, irgendwann genau das zu tun, um exotische Funktionen zu finden, von deren Existenz man nicht einmal zu träumen gewagt hat.

angenehmere und spannendere Tätigkeit und verbringen folglich ihre Zeit lieber mit dem Schreiben von Code als mit der Suche nach bereits geschriebenem. Auch überschätzt man gern die eigenen Fähigkeiten und unterschätzt die Komplexität des Problems. Gerade Aufgaben, die überschaubar wirken, bringen dabei Probleme mit sich, weil auch unerfahrene Programmierer bei ihrer Betrachtung sofort einen Lösungsweg erkennen oder zu erkennen glauben.

Aufmerksame Leser werden in diesen Problemen die Kehrseite von Larry Walls wichtigen Programmiertugenden Faulheit, Ungeduld und Selbstüberschätzung (siehe Kapitel 2) wiedererkennen. Zum Vorteil gereichen diese Eigenschaften Programmierern nämlich nur dann, wenn sie bei den passenden Anlässen zum Einsatz kommen. Wer alle seine Tools ohne vorherige Recherche selber schreibt, dem fehlt schlicht die Zeit, mithilfe seiner Untugenden eines Tages etwas wirklich Nützliches und bisher nicht Dagewesenes hervorzubringen. Denn eine erste funktionierende Version der eigenen Datumsfunktionen, der eigenen Blogsoftware oder des eigenen Zeiterfassungstools ist zwar schnell geschrieben, aber die im weiteren Verlauf überraschend auftauchenden Sonderfälle, Bugs, Erweiterungswünsche und – wenn andere Menschen den Code nutzen – Supportanfragen können eine erstaunliche Menge Lebenszeit verschlingen. Ein erfahrener Softwareentwickler produziert einer gängigen Faustregel zufolge inklusive Nachdenken, Testen, Debugging, Optimierung und Dokumentation ungefähr zehn ausgereifte Zeilen pro Tag. (Bei Buchautoren ist es ähnlich.) Als unerfahrener Entwickler schaffen Sie sicher nicht mehr.

Es gibt für viele immer wiederkehrende Problemfälle inzwischen Standardlösungen, die von mehreren Generationen von Programmierern verfeinert wurden, zum Beispiel Sortieralgorithmen. Dass man an einem Problem arbeitet, das ein spezielles, bisher noch nicht existierendes Sortierverfahren benötigt, ist selbst für mittelgute Programmierer unwahrscheinlich. Nur weil die vom Framework gelieferte Standardlösung nicht absolut passgenau ist, sollte man sich nicht dazu verleiten lassen, etwas Eigenes zu schnitzen, denn man verbringt damit viel Zeit, baut vermutlich Fehler ein und arbeitet gegen das Framework – das heißt, man hat zwar danach einen schönen Sortieralgorithmus für sich, aber die Schwierigkeiten mit der Passgenauigkeit werden an anderer Stelle wieder auftauchen.

Für den Einsatz fertiger Lösungen spricht auch, dass man mit selbstgeschnitzten Werkzeugen zukünftigen Nutzern und Lesern des Codes keine Freude macht. Vielleicht möchten diese hilfsbereiten Leser die Performance der Anwendung verbessern oder nach Fehlern suchen. Wenn dort statt Aufrufen von vertrauten Standardfunktionen eine eigene Lösung auftaucht, erregt der unbekannte Code zu Recht Misstrauen. Die Leser müssen sich nun beispielsweise mit den Details einer Spezial-Sortierfunktion auseinandersetzen, um sich davon zu überzeugen, dass sie fehlerfrei ist und das Problem effizient löst. Hätte der Programmierer hingegen eine Standardfunktion verwendet, könnte sich ein erfahrener Leser diesen Aufwand sparen, denn auf die Implementierung der Stan-

dardfunktion haben so viele Augenpaare gesehen, dass sie mit einiger Sicherheit nicht nur viel weniger Fehler enthält, sondern auch effizient ist.

Der Weg zur Lösung

Wenn man feststellt, dass man immer wieder ähnlichen Code für eine relativ überschaubare Aufgabe selbst schreibt, ist das ein Anzeichen dafür, dass wahrscheinlich bereits eine fertige, wenige Zeichen lange Lösung existiert. Es gibt diverse Möglichkeiten, diese Lösung aufzuspüren:

- Man liest in der Dokumentation der Sprache einen verwandten Bereich durch und hofft auf Querverweise zur gesuchten Funktion.
- Man betrachtet eine Liste aller Funktionen oder der Funktionen eines bestimmten Themenfeldes und hofft darauf, dass das gesuchte Ding einen sprechenden Namen trägt: `array_rand` aus dem oben genannten Beispiel etwa hätte sich in der PHP-Dokumentation unter *php.net* leicht durch Betrachten des Abschnitts »Array Functions« finden lassen.
- Man konsultiert ein Buch, das Standardlösungen auflistet. Die englischen »Cookbooks« bzw. deutschen »Kochbücher« von O'Reilly eignen sich dafür, denn sie sammeln typische Fragestellungen aus der Programmierpraxis und beantworten sie rezeptartig.
- Man wirft das Problem einer Suchmaschine vor und hofft, zum Beispiel bei *stackoverflow.com* dazu eine Frage zu finden, bei deren Beantwortung sich mehrere Autoren gegenseitig mit immer eleganteren Lösungen übertreffen. Auf `array_rand` wäre man zum Beispiel durch eine Suche nach *»how to« »random element« array php* gestoßen.
- Man sucht zum Beispiel auf *github.com* oder *sourceforge.net* in den Beschreibungen von Open Source-Projekten nach einem Projekt in der jeweiligen Sprache, das dieses Problem mit großer Wahrscheinlichkeit auch zu lösen hatte. Dann sieht man nach, wie dessen Autoren die Sache angegangen sind.
- Wenn man etwas mehr Erfahrung gesammelt hat, weiß man auch in einer neuen Sprache, mit welchen Grundfunktionen zu rechnen ist, und braucht dann nur noch deren Namen und die technischen Details herauszufinden. Man muss der Suchmaschine keine mühsame Beschreibung vorlegen wie in unserem Beispiel mit *»how to« »random element« array php*. Stattdessen kann man einfach nach *array_rand in python* suchen, um das Python-Äquivalent zum schon bekannten PHP-Befehl zu finden.

Erst wenn wirklich nirgendwo wiederverwendbarer Code zu entdecken ist, lohnt es sich, sich selber mit der Aufgabe zu befassen. Dabei kann dann auch in einer anderen Sprache geschriebener Quellcode sehr nützlich sein, und sei es nur, weil man nach der Betrachtung die tatsächliche Größe der selbstgestellten Aufgabe besser einschätzen kann.

Mehr zuhören, weniger arbeiten

»Dieser Tage lerne ich unnötige Arbeit vermeiden, indem ich ständig Podcasts über Open Source-Projekte höre. Und zwar lerne ich speziell aus denen, deren Titel mich erst einmal gar nicht interessieren. Damit man das Rad nicht neu erfindet, muss man überhaupt erst mal wissen, dass es das Rad gibt. Ein Überblick über die Open-Source-Landschaft hilft da sehr. Außerdem helfen diese Podcasts, die schädliche ›Open Source ist unbrauchbar, buggy und hässlich‹-Haltung abzulegen, die nur eine weitere Spielart des ›Not Invented Here‹-Problems[2] ist, und zwar, weil man die Projektgründer begeistert über ihre Kinder sprechen hört.

Der englischsprachige Podcast FLOSS weekly (*twit.tv/FLOSS*) stellt regelmäßig Open Source-Projekte vor und lässt Projektbeteiligte über ihre Beweggründe und Designentscheidungen berichten. In einigen Folgen tut das auch der deutschsprachige Podcast Chaosradio Express (*chaosradio.ccc.de/chaosradio_express.html*). In anderen Folgen wird ein Thema breiter behandelt. Programmierern, die sich erstmals mit einer neuen Sprache, Bibliothek oder Technologie befassen, sei hiermit sehr ans Herz gelegt, vorab ein paar Stunden mit Zuhören zu verbringen.«

Jan Bölsche

Bibliotheken

Wenn Sie auf der Suche nach einer spracheigenen Funktion für eine bestimmte Aufgabe nicht fündig geworden sind, dann kann es sein, dass der eigentliche Sprachkern dafür keine Bordmittel bereitstellt. Für etwas komplexere Aufgaben (sich mit einer Datenbank verbinden, XML parsen, Bild- oder Sprachverarbeitung betreiben) werden Sie in der Werkskonfiguration der Sprache Ihrer Wahl vielleicht keine fertigen Werkzeuge finden. Sie sollten sich dann auf die Suche nach geeigneten Bibliotheken machen.

Eine Bibliothek[3] ist eine Sammlung von Funktionen – oder, in objektorientierten Sprachen, Klassen –, die Probleme lösen, die in mehr als nur einem Programm auftauchen. (Richtig, das trifft auf erstaunlich viele Probleme zu.) Der Autor einer Bibliothek gibt sich im Idealfall besonders viel Mühe, die Schnittstelle zu seinem Bibliothekscode für ein breites Publikum verständlich zu gestalten und gut zu dokumentieren. Diese Oberfläche einer Bibliothek wird auch Application Programming Interface (API) genannt. Bei der Gestaltung einer guten API legen ihre Entwickler besonderes Augenmerk auf die konsequente Einhaltung von Konventionen (siehe Kapitel 4), um dem Applikationsprogrammierer durch Vorhersagbarkeit das Leben zu erleichtern.

2 *en.wikipedia.org/wiki/Not_invented_here*.

3 Die Begriffe Bibliothek, Library, Package und Paket nutzen wir hier als Synonyme.

Programmiersprachen werden dem Entwickler nicht nackt vor die Füße geworfen, sondern bringen eine sehr allgemeine Funktionssammlung mit, ohne die es mühsam wäre, irgendein sinnvolles Programm zu schreiben. Eine solche Sammlung heißt Standardbibliothek, Laufzeitbibliothek oder Runtime Library[4]. Das Angenehme an Standardbibliotheken ist, dass man als Programmierer ihre Existenz einfach voraussetzen und ihre Funktionen ebenso selbstverständlich benutzen kann wie die fest eingebauten Befehle der Sprache. Oft ist die Grenze zwischen Sprachkern und Standardbibliothek gar nicht so einfach auszumachen, in der Praxis ist sie auch kaum relevant.

Anders sieht es mit den Bibliotheken aus, die nicht fest mit der Sprache verheiratet sind, denn sie müssen vom Programmierer erst einmal gefunden und dann heruntergeladen und installiert werden. Auch auf dem Rechner des Endanwenders müssen diese Spezialbibliotheken vorhanden sein, es sei denn, Sie beschränken sich auf Webanwendungen, die nur einen Browser erfordern. Wenn Sie solche Bibliotheken einsetzen wollen, müssen Sie sich also Gedanken darüber machen, wie diese Bibliotheken auf Ihren Rechner und gegebenenfalls auf die Ihrer Anwender kommen. (Auch dafür gibt es typischerweise Lösungen, die man verwenden sollte, anstatt sich etwas auszudenken.)

Richtig kompliziert und nervenaufreibend kann es werden, wenn eine Bibliothek wiederum drei andere Bibliotheken voraussetzt und die jeweils noch fünf andere. Diese Abhängigkeiten erreichen oft eine Komplexität, die kaum noch handhabbar ist, weshalb weniger abgebrühte Programmierer irgendwann entnervt aufgeben und doch lieber alles wieder selber machen. Programme, die Ihnen helfen, solche Abhängigkeiten aufzulösen und Bibliotheken zu installieren, heißen Paketmanager oder *package manager* (siehe den Abschnitt »Paketmanager« in Kapitel 20).

Wenn es zur Programmiersprache Ihrer Wahl eine Community gibt, die eine solche paketbasierte Installation von Bibliotheken gebaut hat und ein entsprechendes Onlineverzeichnis pflegt, haben Sie Glück. Denn diese Verzeichnisse sind die komfortabelste Möglichkeit, nach einer Bibliothek für ein vorhandenes Problem zu suchen. Sie sollten daher die erste Anlaufstelle sein. Teilweise bieten diese Kataloge abonnierbare RSS-Feeds, die über Neuerscheinungen informieren, und manche haben ein Bewertungssystem oder ein anderes Verfahren zur Qualitätssicherung. Einige solcher Anlaufstellen finden Sie in Tabelle 19-1.

Tabelle 19-1: Bibliotheksverzeichnisse

Sprache	Bibliotheksverzeichnis	Webadresse
JavaScript	JavaScript Libraries	*javascriptlibraries.com*
Python	Python Package Index	*pypi.python.org*
Perl	Comprehensive Perl Archive Network	*cpan.org*

4 Typischerweise beinhalten diese Bibliotheken so grundlegende Funktionen wie Dateioperationen, Texteinund ausgaben, mathematische Funktionen und Funktionen zum Umgang mit Listen, Arrays und Strings.

Tabelle 19-1: Bibliotheksverzeichnisse (Fortsetzung)

Sprache	Bibliotheksverzeichnis	Webadresse
R	Comprehensive R Archive Network	*cran.r-project.org*
Ruby	Ruby Gems	*rubygems.org*
PHP	PECL (PHP Extension Community Library)	*pecl.php.net*
PHP	PEAR (PHP Extension and Application Repository)	*pear.php.net*
Node.js	Node Packaged Modules	*npmjs.org*

Für Java und C++ fehlt so ein praktisches Verzeichnis leider. *javascriptlibraries.com* enthält nur die wichtigsten JavaScript-Bibliotheken. Für C++-Programmierer sei aber die Bibliothekensammlung *boost.org* erwähnt. Die ist zwar weniger guten Programmierern aufgrund ihrer Komplexität nicht eben leicht zugänglich, aber C++-Programmierer sind diesen Kummer ja gewohnt.

Außer diesen sprachspezifischen Verzeichnissen gibt es Internetdienste wie *github.com*, *bitbucket.org* und *sourceforge.net*, auf deren Servern alle möglichen Open Source-Projekte ihren Code zum Download bereitstellen.

Bindings

Oft basieren übrigens die verschiedenen sprachspezifischen Lösungen alle auf ein und derselben Bibliothek, die meist in C geschrieben ist. Sie bestehen dann eigentlich nur aus einem kleinen Übersetzungsteil, der zwischen den unterschiedlichen Sprachwelten vermittelt. Ein solcher Übersetzungsteil wird Binding genannt. Wenn Sie z.B. den Satz hören, »Wir verwenden das PHP-Binding von open-ssl«, dann heißt das, dass in einem Projekt die (in C geschriebene) Kryptobibliothek open-ssl zum Einsatz kommt, die durch ein entsprechendes PHP-Modul in die Welt von PHP eingeklinkt wurde.

Für quasi alle größeren Aufgabenbereiche gibt es schon kostenlose, frei verfügbare Libraries. Allerdings meist nicht nur eine, sondern zwischen zwei und zehn. Das stellt den Programmierer vor ein Auswahlproblem und kostet Zeit. Faustregel: Die am weitesten verbreitete nehmen, oder die mit der besten Website (ein Zeichen dafür, dass die Entwickler Wert auf Zugänglichkeit legen), oder die mit der besten Dokumentation und verständlichsten API.

Und beachten Sie die Lizenz. Es gibt Lizenzen wie die GPL, die Sie zwingen, Ihren gesamten Programmcode ebenfalls unter derselben Lizenz freizugeben, falls Sie irgendein Stück Fremdcode einbinden, das unter dieser Lizenz steht. Die Affero-GPL (AGPL) geht sogar noch einen Schritt weiter und zwingt Sie, den Code Ihres Webservice unter der AGPL zu veröffentlichen, wenn Sie AGPL-lizensierte Bibliotheken verwenden. Bibliotheken wer-

den zwar meist unter weniger restriktiven Lizenzen wie der BSD-, der MIT- oder der LGPL-Lizenz veröffentlicht, aber schauen Sie dennoch genau hin, falls Sie nicht sowieso Open Source-Programme schreiben.

Umgang mit Fremdcode

Wenn Sie Codeschnipsel zusammenkopieren oder Ihr Codingstil durch die Lektüre von Fremdcode beeinflusst wird, ergibt sich ein heterogenes Bild im entstehenden Code. Das ist im Grunde unvermeidbar; schlechte Programmierer müssen es einfach hinnehmen – schon weil es im Laufe ihrer Entwicklung zu weniger schlechten Programmierern sowieso passieren wird. Am besten akzeptiert man diese Tatsache und macht die Grenzen von Eigen- und Fremdcode möglichst gut erkennbar, indem man den Fremdcode in separate Dateien auslagert. Den fremden Code mühsam den eigenen Konventionen (siehe Kapitel 4) anzupassen, hat ohnehin nur Nachteile: Erstens wird man die Lage voraussichtlich eher verschlimmern, zweitens verzettelt man sich, und drittens ist das Ergebnis nicht wartbar bei Updates des Fremdcodes. Stattdessen kann man in Erwägung ziehen, komplizierte Dinge schön zu verpacken, indem man etwa außen um den Aufruf einer Bibliotheksfunktion herum eine eigene Funktion bastelt, die eine stark vereinfachte Parameterliste hat und all die Funktionalität weglässt, die man im eigenen Projekt nicht braucht. Man spricht dann von *code wrapping*.

Kleinere Stücke Fremdcode kann man jederzeit übernehmen, auch wenn man sie noch nicht ganz versteht. Es ist eine der schnellsten und elegantesten Arten, dazuzulernen. Ein bisschen allerdings sollte man schon begreifen, was der fremde Code da tut. Wenn nicht, ist die Gefahr groß, dass er ganz andere Aus- und Nebenwirkungen als die beabsichtigten hat. Bei umfangreicheren Übernahmen hilft es, die URL der Quelle als Kommentar zu vermerken. Man kann sich dann später noch einmal die Erklärungen auf der Seite ansehen, und manchmal gibt es sogar Updates. Diese Praxis beugt außerdem dem Vorwurf vor, man würde sich mit fremden Federn schmücken. Die Quelle zu vermerken, ist also selbst dann eine gute Idee, wenn es von der Lizenz nicht gefordert wird.

Beliebte Fehler im Umgang mit Fremdcode

Verändern Sie keinen Bibliothekscode in der lokalen Kopie. Sie können den fremden Code danach nie wieder updaten. Was Sie da angelegt haben, ist ein *Fork*, auch wenn Sie von Forks gar nichts wissen und nie etwas wissen wollten.

Fork

Als Fork (im Sinne von Gabelung) bezeichnet man ein Softwareprojekt, das irgendwann einmal durch Änderungen an einem Originalprojekt entstanden ist. Weil diese Änderungen nicht in den Entwicklungsstrang des Originalprojekts zurückgeflossen sind (dafür kann es viele Gründe sozialer, politischer oder technischer Natur geben), haben sich die beiden Projekte

im Laufe der Zeit immer weiter von einander entfernt und streben unterschiedlichen Zielen entgegen. Verbesserungen und Fehlerbehebungen an einem dieser Projekte in das andere zu bekommen, wird daher immer aufwendiger. Ein Beispiel sind etwa die verschiedenen Derivate des Betriebssystems BSD: OpenBSD, FreeBSD und NetBSD.

Verwenden Sie fremden Code nicht stillschweigend, wenn dessen Autor sich ausdrücklich eine Namensnennung wünscht. Das könnten Sie aus verschiedenen Gründen tun wollen, von ästhetischen Bedenken (»wo soll ich den Namen denn jetzt unterbringen«) über persönliche Eitelkeit (»die Leute denken ja, ich komm nicht selber klar«) bis hin zu Kundenwünschen. Die einzige korrekte Lösung aber lautet: Wenn man nicht willens oder in der Lage ist, den Autor zu nennen, darf man solchen Code nicht verwenden.

Verwenden Sie keinen Fremdcode zu anderen als den vorgegebenen Bedingungen. Wer Fremdcode übernimmt, muss nachsehen, unter welcher Lizenz dieser Code steht, und sicherstellen, dass die genannten Bedingungen erfüllt werden. Alles andere stellt eine Verletzung des Urheberrechts des Codeautors dar, wofür sich der Gesetzgeber teilweise recht drakonische Strafen hat einfallen lassen. Außerdem gehört es sich nicht.

Was man nicht selbst zu machen braucht

Die folgenden Aufgaben brauchen Sie nicht selbst zu erledigen. Es passiert aber auch nichts Schlimmes, wenn Sie es tun. Sie machen sich lediglich selbst das Leben etwas schwerer.

Eine Funktion schreiben, die aus Kleinbuchstaben Großbuchstaben macht

Sie bekommen das mit Sicherheit für Deutsch und Englisch korrekt hin. Vielleicht sogar unter Berücksichtigung der besonderen Eigenschaft von »ß«, die Zeichenkette zu verlängern, wenn man es in »ss« umwandelt. Anderen ist das aber bereits für wesentlich mehr Sprachen gelungen. Sie glauben jetzt, dass andere Sprachen Sie nicht zu interessieren brauchen, und übersehen dabei, dass zum Beispiel Orts- und Familiennamen manchmal auch lustige Sonderzeichen enthalten.

Eine Suchfunktion für die eigene Website

Die Suche selbst mag schnell geschrieben sein. Dafür zu sorgen, dass sie Eingaben in den üblichen Formaten verarbeitet und die Ergebnisse in benutzerfreundlicher Form darstellt, ist mehr Arbeit und wird deshalb häufig vernachlässigt. Die großen Suchmaschinen bieten Anleitungen dafür, wie man mit geringem Aufwand z. B. eine Google-Suche in die eigene Website einbinden kann. Und auch für die meisten anderen Anwendungsfälle gibt es fertige Lösungen. Lucene und Lucy, beides Open Source-Projekte der Apache Foundation, sind beispielsweise Bibliotheken für eine Volltextsuchfunktion für Java, C, Perl und Ruby.

Eigene Parser für Dateiformate oder URLs

»Comma-separated values, wie schwer kann das schon sein, es sind halt Werte mit Kommas dazwischen!«, denken Sie, und werfen mit `String.Split(',')` oder `explode(",", $string)` die Kommas weg. Tatsächlich gibt es selbst bei so einfachen Formaten wie CSV und JSON überraschend viele Möglichkeiten, sich in Schwierigkeiten zu bringen: Kommas können escaped sein oder im Inneren von Zitaten stehen, dasselbe gilt für Anführungszeichen; Werte können sich über mehrere Zeilen erstrecken ... von komplizierteren Formaten wie XML wollen wir gar nicht erst anfangen. Auch die Parameter aus URLs sollten Sie aus ähnlichen Gründen nicht von Hand herausfiletieren.

Dasselbe gilt für die umgekehrte Richtung: Schreiben Sie keinen eigenen Serializer (eine Funktion, die Variablen oder Objekte in speicher- bzw. verschickbare Form bringt). Nutzen Sie die fertigen Lösungen, die es reichlich gibt, dann brauchen Sie sich nicht selbst um Escaping, richtige Klammersetzung und andere lästige Details zu kümmern.

Kommunikation mit SQL-Datenbanken

Für fast jede Kombination aus Programmiersprache und Datenbank gibt es Module, die den Zustand eines Objektes oder einer Struktur in einer Datenbanktabelle ablegen, beziehungsweise einen bereits abgelegten Zustand wiederherstellen können, sogenannte »object-relational Mapping Libraries« (ORM). Die Idee dahinter ist, dass der Programmierer sich nicht mit der Erzeugung von SQL-Befehlen auseinandersetzen muss. Denn genau das ist erstens komplizierter, als man denkt, zweitens für jede Datenbank subtil anders und drittens der Grund für eine ganze Klasse von Sicherheitslücken (siehe dazu den Abschnitt »SQL Injection und XSS – die Gefahren in User-Content« in Kapitel 25)

Regular Expressions für gängige Aufgaben selbst konstruieren

Wenn auch nur der Hauch eines Verdachts besteht, dass jemand anders dasselbe Problem bereits gelöst hat, spart man Zeit und Debugging-Aufwand, indem man die fertige Lösung einsetzt. Beim Finden hilft z. B. *regexlib.com*.

Funktionen schreiben, die dafür sorgen, dass irgendetwas zu bestimmten Tages-, Wochen- oder Jahreszeiten geschieht

Auch diese Aufgabe wirkt auf den ersten Blick einfach und erweist sich schon kurze Zeit später als überraschend reich an Fallstricken. Vorausschauendere Menschen machen deshalb von den eingebauten Fähigkeiten ihrer jeweiligen Betriebssysteme Gebrauch: Unter Windows gibt es den *Windows Task Scheduler*, unter Unix (und damit auch am Mac) nutzt man das Unix-Tool `cron` (siehe Kapitel 22). Wenn die Zeitschaltuhr wirklich in den Code eingebaut werden muss, gibt es auch dafür Bibliotheken. Man findet sie mit einer Suche nach *scheduling library* in Kombination mit dem Namen der jeweiligen Programmiersprache.

Nichttriviale Mathematik- oder Physikalgorithmen

Wenn Sie nicht gerade Mathematiker oder Physiker sind, sollten sie alles, was wesentlich über Ihre Erinnerung an den Schulunterricht hinausgeht, anderen überlassen. Solche Algorithmen sind nicht nur schwer fehlerlos hinzubekommen, sondern auch schwer zu testen. Es gibt Bibliotheken dafür, die von freundlichen Fachleuten durchgesehen und korrigiert wurden. Außerdem sind deren Lösungen wahrscheinlich um ein Vielfaches schneller und effizienter als Ihre eigene Version. Wenn Sie schon wissen, dass Sie in einem Projekt mit mathematisch-statistischen Problemen konfrontiert sind, dann verwenden Sie entsprechende Programme wie SAS oder eine Sprache wie R.

Ein Tool schreiben, um die Geschwindigkeit von Code zu messen

Auch diese Aufgabe ist komplexer, als sie zunächst aussieht. Laufen andere Prozesse im Hintergrund? Wie viel Prozessorzeit widmet der Rechner der Ausführung? Das ist vor allem (aber nicht nur) dann ein Thema, wenn man sich einen Server mit anderen Nutzern teilt. Es gibt für jede Programmiersprache fertige Lösungen, für JavaScript sind das Firebug (Firefox) oder Speedtracer (Chrome), Java hat `jvisualvm`, Python z. B. das `timeit`-Modul, in Ruby verwendet man `Benchmark`. Suchbegriffe sind *timing code* oder *code profiler*. Wer Datenbanken verwendet, sollte sich irgendwann einmal mit `explain` beschäftigen. Dieser SQL-Befehl gibt interessante interne Details darüber aus, wie eine Abfrage ausgeführt wurde. Mit etwas Übung können Sie daraus erschließen, ob die Datenbank eine Query optimieren konnte, ob Indizes verwendet werden konnten und wie Sie die Abfrage möglicherweise noch schneller machen können.

Ein Template-System zum Erzeugen von Websites schreiben

Es gibt genügend fertige Tools, die eine Suche nach `template engine` in Kombination mit der jeweiligen Sprache zutage fördert.

Einen eigenen Texteditor für HTML-Eingabefelder oder einen WYSIWYG-Editor, der im Browser läuft

Wie immer ist es leicht, die ersten 80 % der Funktionalität hinzubekommen. Für die letzten 20 % brauchen (im Fall der WYSIWYG-Editoren) selbst erfahrene Programmiererteams Jahre. Und auch bei den HTML-Eingabefeldern lohnt sich die Mühe angesichts der zahlreichen fertigen Lösungen nicht.

Mouseover-Code und andere gebräuchliche Website-Verschönerungen

Es ist sehr wahrscheinlich, dass um die 99 % dessen, was Sie sich an Ajax-Funktionen wünschen, bereits in einer fertigen, fehlerfreien Version existiert. Fremdcode spart hier nicht nur Zeit und Arbeit, sondern bewahrt einen auch davor, sich bei Google unbeliebt zu machen. Google-Mitarbeiter Matt Cutts rät ausdrücklich davon ab, Mouseover-Code

selbst zu schreiben, weil man dabei leicht versehentlich das Misstrauen der Google-Algorithmen erweckt, die Seiten mit verstecktem Spamtext aus dem Suchindex verbannen. *script.aculo.us* ist eine gute Anlaufstelle für fertigen Ajax-Zierrat.

Eigene Internationalisierungstools

Die Texte Ihrer Website sollen in unterschiedlichen Sprachen angezeigt werden. Diesen Wunsch hatten schon viele Menschen vor Ihnen, weshalb es auch fertige Hilfsmittel dafür gibt, z. B. die `gettext`-Utilities mit Bindings für alle gängigen Sprachen. Um weitere geeignete Bibliotheken zu finden, suchen Sie nach *internationalization*, *localization*, *i18n* in Kombination mit den Namen Ihrer bevorzugten Sprache.

Eigene Logging-Tools

Das gilt sowohl für Error-Logging als auch für Zugriffe auf Webseiten, Verfolgung von Nutzeraktionen und die Auswertung von Logfiles. `log4j` ist eine Logging-Bibliothek (siehe den Abschnitt »Logging« in Kapitel 13), die als log4js (JavaScript), log4r (Ruby), Log4net (.NET) in andere Sprachen übersetzt wurde. (In Python wurde sie gleich integriert und heißt dort einfach `logging`.) Um von `log4j` oder ähnlichen Paketen geschriebene Logs analysieren zu können, gibt es beispielsweise die Software *Apache Chainsaw*, die leider nicht mehr weiterentwickelt wird – allerdings kann man kurze Logfiles auch einfach im Texteditor lesen und analysieren.

Eine eigene Auszeichnungssprache für Text erfinden

Wenn Sie ein einfaches und laienfreundliches Format für die Eingabe von Text brauchen, der später automatisch z. B. in HTML umgewandelt werden soll, denken Sie sich bitte nicht selbst aus, wie kursiver oder fetter Text oder Listen gekennzeichnet werden sollen. Es gibt dafür zwar keine einheitlichen Standards, aber doch übliche Lösungen für bestimmte Anwendungsbereiche. So haben Wikis ihre eigenen Konventionen, in Foren kommt ein Format namens *BBCode* zum Einsatz, vielleicht ist auch *Markdown* oder *Textile* das, was Sie suchen. Der Wikipedia-Eintrag »Markup« bietet einen kurzen Überblick. Der Vorteil beim Einsatz einer gängigen Auszeichnungssprache: Es gibt fertige Bibliotheken für die Umwandlung aus vielen Formaten und in viele Formate. Wenn Ihnen also nach ein paar Wochen einfällt, dass Sie das Ergebnis statt als HTML doch lieber als PDF hätten, verursacht das wenig Arbeit.

Eigene Konventionen erfinden

Wenn Sie mit anderen zusammenarbeiten, verzichten Sie auf das Festlegen ganz neuer Codestandards. Es gibt genügend gebräuchliche Vorgaben auf der Welt, und die Einigung mit anderen ist schon mühsam genug, ohne dass man eigene Erweiterungen verteidigen muss (siehe auch Kapitel 4).

Was man auf keinen Fall selbst machen sollte

Die hier versammelten schlechten Ideen haben schmerzhaftere Folgen als die aus dem vorigen Abschnitt. Manche davon werden nicht nur Ihnen, sondern auch Ihren Nutzern Ärger und Kosten bescheren.

Datumsberechnungen anstellen

Unterschiedlich lange Monate. Zeitzonen. Sommerzeit. Die Nummerierung von Kalenderwochen. Schaltjahre. Schaltsekunden! Über all das haben sich viele andere Menschen ausführlich den Kopf zerbrochen und das Ergebnis in Bibliotheken niedergelegt. Machen Sie davon Gebrauch, sonst ergeht es Ihnen wie Jan Bölsche (siehe den Kasten »Mach es nicht selbst, sonst gibt es morgen schlechtes Wetter!«). Wenn Sie diese Regel beherzigen, haben Sie den Amazon- und Apple-Programmierern schon etwas voraus: Bei Amazon tragen am 29. Februar eingestellte Kindle-Bücher das Veröffentlichungsdatum 28.2., die Verkäufe vom 29. werden hingegen auf den ersten März gebucht. Apples Time Machine speichert zwar die Backupdaten vom 29. Februar, das Datum wird in der Benutzeroberfläche aber nicht angezeigt.

Mach's nicht selbst, sonst gibt es morgen schlechtes Wetter!

»Es regnet hier nun schon drei Tage durch.« Der Projektleiter am Telefon klingt etwas besorgt. Er steht unter einem künstlichen Himmel aus LEDs und blickt in die finsteren Wolken über ihm, die sich eigentlich zeitgleich mit den realen Wolken über dem Gebäude vor ein paar Tagen hätten verziehen sollen. Und zwar vollautomatisch, als Reaktion auf eine Änderung in einer Textdatei auf dem Server des Meteorologischen Instituts Berlin.

Seit gut zwei Jahren hatte das vollkommen reibungslos funktioniert: Ein wissenschaftlicher Mitarbeiter im Institut trägt Bewölkungsgrad und die Art des Niederschlags in eine Textdatei ein, die wird von meiner Software alle 10 Minuten runtergeladen und geparst und dient dann als Input für die Erzeugung von Wolken auf dem riesigen Display unter der Decke des Wintergartens.

Während ich aus dem Telefon Regenprasseln höre, sehe ich im Logfile der Applikation nach und muss feststellen, dass sie erst in ungefähr 2 Milliarden Jahren wieder plant, sich beim Institut nach der aktuellen Wettersituation zu erkundigen.

Dabei war ich mir ganz sicher, dass die selbstgeschriebene Datumsklasse nach Jahren des ständigen Einsatzes und der Pflege nun aber wirklich fehlerfrei ist.

Jan Bölsche

Kryptografie selbst implementieren

Ohne Ausnahme eine schlechte Idee, auch wenn Ihr Plan über »ich XOR das einfach mal mit dem Namen meiner Katze« hinausgeht (siehe Kapitel 25). Kryptografie ist – ähnlich wie Atomenergie – ein Bereich, den man unbedingt Experten überlassen muss, da alle weniger Bewanderten ganz sicher die Konsequenzen ihres Handelns nicht überblicken. (Manche sagen, dass das zumindest im Falle der Atomenergie auch bei Experten nicht der Fall sei.) Bedenken Sie das nach dem Sicherheitsberater Bruce Schneier benannte »Schneier's Law«: »Es ist für alle Menschen ganz einfach, sich Sicherheitsverfahren auszudenken, die sie selbst nicht knacken können.«

Um Kryptografie handelt es sich übrigens nicht nur, wenn Sie vom Geheimdienst dafür bezahlt werden, sondern immer dann, wenn Passwörter im Spiel sind. Falls Sie denken, »Ach, so geheim sind die Daten nun auch nicht, da brauch ich doch keine echte Kryptografie«, dann stellen Sie sich vor, wie es wäre, alle Daten Ihrer Nutzer im Klartext abzulegen oder zu übertragen. Wenn Ihnen dieser Gedanke unangenehm ist, dann nehmen Sie »echte Kryptografie«, eine Zwischenlösung gibt es nicht. Das bedeutet unter anderem, dass Sie am besten keine eigenen Loginseiten bauen. Erstens ist es sicherer, zweitens gibt es fertige Lösungen und drittens kostet Sie der Umgang mit vergesslichen Nutzern weniger Nerven (siehe Kasten).

Passwort: Alzh31m3r

Weil das Riesenmaschine-Blog historisch aus einer kleinen ›News‹-Abteilung einer Website hervorgegangen ist, auf die nur drei Leute Zugriff hatten, und weil ich keine Ahnung von Authentifizierung habe, ist die Zugangssicherung einfach ein `.htaccess`-Verzeichnisschutz. Das hat vermutlich viele Nachteile, von denen ich bis heute nichts weiß, und einen auch für mich unübersehbaren: Jeder der zeitweise über 50 Zugangsberechtigten vergisst regelmäßig sein Passwort, und es gibt natürlich keinen Mechanismus, mit dem die Nutzer ihr eigenes Passwort ändern können. Das geht nur mit einer Mail an mich, und da ich mir die Passwörter natürlich auch nicht merke, muss ich mir ein neues ausdenken, das der Empfänger dann bis zur nächsten Woche wieder vergisst.

Kathrin

Ein eigenes Bugtracking-System entwickeln

Stack Overflow-Nutzer Constantin Veretennicov hat Zahlen über die existierenden Systeme zusammengetragen (Stand Ende 2012):

Trac: 44.000 Codezeilen, 10 Personenjahre, $ 577.003 Entwicklungskosten

Bugzilla: 54.000 Codezeilen, 13 Personenjahre, $ 714.437 Entwicklungskosten

Redmine: 171.000 Codezeilen, 44 Personenjahre, $ 2.400.723 Entwicklungskosten

Mantis: 182.000 Codezeilen, 47 Personenjahre, $ 2.562.978 Entwicklungskosten

Wenn Sie mit den Stärken und Schwächen der existierenden Systeme vertraut sind und gerade ein paar Jahrzehnte Zeit übrig haben, lassen Sie sich bitte nicht durch uns von Ihrem Plan abbringen. Die heutigen Bugtracker sind sicher noch nicht der Weisheit letzter Schluss. Wenn nicht: Finden Sie sich damit ab, dass die existierenden Bugtracker nicht 100 % des Funktionsumfangs abdecken, der Ihnen vorschwebt, und genießen Sie 10 bis 47 Personenjahre zusätzliche Freizeit.

Ein eigenes Wiki entwickeln

Hier gilt das Gleiche wie beim Bugtracking-System.

Eigene Blogsoftware entwickeln

Hier gilt das Gleiche wie beim Bugtracking-System.

Eigene Dateiformate erfinden

Weil man gerade keine Lust hat, sich mit den Feinheiten von XML oder JSON zu befassen, schludert man schnell so was Ähnliches hin. Das ist an sich schon fahrlässig genug, aber im ungünstigsten Fall sieht das erfundene Dateiformat seinem Vorbild so ähnlich, dass künftige Leser (darunter man selbst) es für das Original halten und in der Folge viel Zeit mit Wundern und Haareraufen verbringen werden.

Code schreiben, der prüft, ob ein String eine URL oder eine Mailadresse ist

Wer das richtig machen will, braucht entweder eine Bibliothek oder eine Regular Expression von überraschender Länge und Komplexität – und wer es falsch macht, ärgert seine Nutzer, deren völlig legitime Adressen abgelehnt werden. Die vorhandenen Lösungen sind evolutionär aus der Zusammenarbeit vieler Menschen und dem wiederholten Scheitern an Sonderfällen entstanden. Dass jemand im Alleingang dieses Ergebnis reproduziert, ist ungefähr so wahrscheinlich wie die spontane Entstehung eines Gürteltiers aus dem Urschlamm. Nur zur Veranschaulichung sehen Sie hier einen solchen Lösungsvorschlag:

```
\b(([\w-]+://?|www[.])[^\s()<>]+(?:\([\w\d]+\)|([^[:punct:]\s]|/)))
```

(Quelle: *daringfireball.net/2009/11/liberal_regex_for_matching_urls*, hier eine Erklärung mit Verbesserungsvorschlägen: *alanstorm.com/url_regex_explained*)

Hier ein anderer Lösungsvorschlag:

```
/^(https?):\/\/((?:[a-z0-9.\-]|%[0-9A-F]{2}){3,})(?::(\d+))?((?:\/(?:[a-z0-9\-._
~!$&'()+,;=:@]|%[0-9A-F]{2}))(?:\?((?:[a-z0-9\-._~!$&'()+,;=:\/?@]|%[0-9A-F]{2})))?(?:
#((?:[a-z0-9\-._~!$&'()+,;=:\/?@]|%[0-9A-F]{2})*))?$/i
```

(*snipplr.com/view/6889/regular-expressions-for-uri-validationparsing/*)

Noch besser als der Einsatz fertiger Regular Expressions: Investieren Sie ein paar Minuten in die Suche nach *email validation library* in Kombination mit der jeweiligen Sprache.

Greifen Sie auf Regular Expressions nur dann zurück, wenn diese Suche kein brauchbares Ergebnis liefert.

Mithilfe von Regular Expressions HTML-Tags aus einem String entfernen

Dieses Verfahren übt eine unwiderstehliche Anziehungskraft auf weniger erfahrene Programmierer aus. Aber HTML ist komplexer, als man auf den ersten Blick annehmen möchte, und für jede funktionierende Regular Expression[5] gibt es mindestens einen Sonderfall, der sie scheitern lässt – und zwar an völlig legitimem HTML. Chuck Norris kann HTML mit Regular Expressions parsen. Jeder andere sollte stattdessen nach einer für seine Sprache zuständigen Bibliothek googeln (Suchbegriffe: »*HTML Parser*«, »*HTML Sanitizer*«, »*HTML Purifier*« oder *parse html library*). Das spart viel Arbeit sowohl beim Ausdenken als auch beim jahrelangen Korrigieren komplizierter Regular Expressions.[6]

Escaping-Routinen, um SQL, JavaScript oder XSS-Code zu entfernen

Vor allem, aber nicht nur im Web ist es gelegentlich nötig, Steuerzeichen einer Sprache zu maskieren, um Programmstücke in dieser Sprache verarbeiten zu können. Wenn Sie HTML-Beispiele auf einer Webseite darstellen wollen, können Sie nicht einfach das Beispiel-HTML irgendwo einfügen, sondern müssen die Zeichen < und > umcodieren. Escaping von aktiven Inhalten ist notorisch schwierig und es gibt in jeder Sprache Funktionen oder Bibliotheken dafür.

Eigene Lizenzen und Verträge formulieren

Auch hier gibt es fertige Lösungen, über die Fachleute lange nachgedacht haben. Ihre Verwendung erspart Urhebern wie Nutzern viel Arbeit (und potenziellen Ärger). Suchen Sie sich bei »Creative Commons« die richtige Lizenz, wenn Sie Fotos oder Texte online stellen wollen. Und wenn Sie Verträge brauchen, dann beauftragen Sie einen Anwalt.

Leider sind auch offizielle und verbreitete Lösungen nicht immer richtig. Alle guten Ratschläge in diesem Kapitel gelten deshalb nur für schlechte bis mittelmäßige Programmierer uneingeschränkt. Für sie ist die Entscheidung einfach, denn der Code anderer Menschen wird mit großer Sicherheit besser, schneller und fehlerärmer sein als der eigene, und man kann ihn bedenkenlos übernehmen. Trotzdem lohnt es sich manchmal auch für Anfänger, bestimmte Dinge selbst zu schreiben – vorausgesetzt, man kennt die Grenzen dieser Herangehensweise und weiß, worauf man sich einlässt. Das ist leider selten der Fall.

---

5 Die sieht dann zum Beispiel so aus: `<(?:"[^"]*"['"]*|'[^']*'['"]*|[^'">])+>`. Der Autor, Stack Overflow-Nutzer itsadok, warnt: »Diese Regex funktioniert nicht bei CDATA-Blocks, Kommentaren, Script- und Style-Elementen. Die gute Nachricht: Das alles kann man mithilfe einer Regex entfernen.«

6 Zwei Standardtexte zum Thema: *www.codinghorror.com/blog/2009/11/parsing-html-the-cthulhu-way.html* und *oubliette.alpha-geek.com/2004/01/12/bring_me_your_regexs_i_will_create_html_to_break_them*.

Das Netz ist nicht arm an Diskussionen über die Frage, wann es richtig ist, auf eigene Lösungen statt auf die Wiederverwendung vorhandener Tools und Bibliotheken zu setzen. Auch herrscht kein Mangel an Auskünften erfahrener Programmierer, die sich schon einmal oder mehrmals für das Selberschreiben entschieden haben und später zu dem Schluss gekommen sind, dass eine fertige Lösung letztlich auch gereicht und ihnen viel Zeit und Mühe erspart hätte. Eventuell handelt es sich um eine von diesen Erfahrungen, die jeder für sich selbst machen muss. Wenn Sie unsere Ratschläge ignorieren und sich für das Selberschreiben entscheiden, werden Sie hinterher zumindest wissen, was die grundsätzlichen Herausforderungen rund um Internationalisierung oder Datumsfunktionen sind. Das erleichtert Ihnen – einige Wochen, Monate oder Jahre später, nach der Entscheidung, doch auf eine fertige Lösung zu setzen – die Einschätzung des Problems und das Verständnis des Fremdcodes, und es fördert den Respekt vor der Arbeit und den Überlegungen, die in diese Lösung eingeflossen sind.

Drei Lösungen für ein Problem

Kathrin: »Die Person, die ich früher einmal war, hat eine Blog-Engine geschrieben, die die einzelnen Beiträge als XML-Dateien speichert (mit einer Begründung so ähnlich wie ›Eine Textdatei kann ich bei Bedarf einfach anfassen und ändern, einen Datenbankeintrag nicht‹). Zum Umgang mit diesen Dateien schrieb sie einen XML-›Parser‹, der beim Lesen kurzerhand alles wegwirft, was in spitzen Klammern steht, und beim Schreiben wiederum spitze Klammern an die Zeilen montiert. Ein Kommentar im Code vermerkt ›weil der XML-Parser von PHP so scheiße ist‹. Heute bin ich versucht, ein ›Hier irrt die Verfasserin‹ hinzuzufügen. Selbst wenn der PHP-Parser wirklich nichts taugte, wäre ich kaum in der Lage gewesen, das zu diagnostizieren.«

Johannes: »Zu der Zeit, als das Internet noch als Datenautobahn bekannt war und Wikipedia noch nicht erfunden, hatte ich eine Online-Enzyklopädie übernommen, um sie technisch und inhaltlich weiterzuentwickeln. Die erste Version war ein in C geschriebenes Programm ohne Web-Frontend, das Text-Dateien mit ein paar TeX-ähnlichen Formathinweisen in HTML übersetzte.

Das stellte mich vor gewisse Probleme, als ich die Site übernahm: schnell `configure && make` eingetippt, um auf dem neuen Server das Konverterprogramm zum Laufen zu kriegen. Leider war das Ergebnis eine höchst unerfreuliche Liste von Fehlern, weil diverse Bibliotheken nicht mehr kompatibel waren.

Da sich die Welt in den letzten fünf Jahren auch weiterentwickelt hatte, schien es eine hervorragende Gelegenheit, das Konvertierungsprogramm nicht nur neu zu erfinden, sondern die Daten gleich in eine XML-Beschreibung umzuwandeln. Man kann die Begeisterung, mit der die Stadtplaner der Nachkriegszeit die etwas lädierten deutschen Städte endgültig autogerecht verwüsteten, erst nachvollziehen, wenn man sich einmal eine XML-Beschreibung für die Speicherung strukturierter Dateien ausgedacht hat. Herrliche unbeschriebene Blätter, keine Fesseln durch Rückwärtskompatibilität, der Kreativität sind keine Grenzen gesetzt.

Leider waren die Server der damaligen Zeit für die Echtzeitverarbeitung von großen XML-Dateien nicht leistungsfähig genug, und die XML-Parser noch weit vom heutigen Optimierungsgrad entfernt. Um ein Web-Frontend realisieren zu können, verlegte ich mich also darauf, die in XML gehaltenen Daten mit ein paar Regular Expressions in HTML zu verwandeln und die Benutzereingaben auf gleichem Weg wieder zwischen Tags einzutüten. Das erzeugte eine Menge hübscher und schwer zu findender Fehler, die jeden validierenden XML-Parser dazu veranlassten, meine Daten mit gerümpfter Nase abzulehnen. Vor die Wahl gestellt, einen echten XML-Parser und -Generator zu schreiben oder das Problem irgendwie einzudämmen, entschied ich mich zum Glück dazu, den Datenbestand nachts per cron von einem richtigen Parser durchsortieren und die faulen Fische aussortieren zu lassen. Die gefundenen Fehler mussten per Hand ausgebügelt werden, was nicht sehr professionell war, aber leidlich funktionierte. Und das selbstgehäkelte XML war dann doch so brauchbar, dass man es nach HTML oder LaTeX für die Erzeugung von PDF übersetzen konnte.

Im Rückblick wäre es möglicherweise einfacher gewesen, das simple Datenformat der ersten Generation weiter zu verwenden – von den gesamten Zusatzfeatures, die ich eingeführt habe, kam nur der allerkleinste Teil jemals zum Einsatz.«

Jan Bölsche: »Für ein Projekt, das die beiden äußersten, einander feindselig gegenüberstehenden Enden des IT-Branchenspektrums, namentlich Finanzbuchhaltungssoftware und Computerspiel, einander näherbringen soll, brauchte ich eine Möglichkeit, Kontoauszüge zu speichern. Weil diese Aufgabe nicht zeitkritisch ist (der Vorgang wird durch die Kommunikation mit dem Bankserver dominiert, hier ist der Flaschenhals) und weil ich in dieser frühen Phase des Projektes ein einfach zu handhabendes und menschenlesbares Format haben wollte, entschied ich mich für XML. Meine Lieblingssprache war seinerzeit Python, also las ich ein paar Artikel und Blog-Postings über XML-Parsing in Python und sah mir die APIs der verschiedenen XML-Module an, die mit der Python-Distribution ausgeliefert werden. Auf Wikipedia las ich über die grundlegenden Unterschiede zwischen DOM- und SAX-Parsern. Letztendlich entschied ich mich dafür, jeweils die Daten für einen Monat in einer separaten XML-Datei zu speichern. Da diese Dateien nicht besonders groß werden, kann ich sie in einem Rutsch mit einem DOM-Parser lesen, was deutlich unkomplizierter ist, als auf die Parsing-Events eines SAX-Parsers zu reagieren. Am besten hat mir die API von ›ElementTree‹ gefallen, weil diese Bibliothek, obwohl damals noch nicht Teil der Python-Distribution, für meinen Geschmack am ehesten den Grundsatz verfolgt, einfache Lösungen für einfache Probleme bereitzustellen. Eine Liste aller Transaktionen eines Monats liefert beispielsweise dieser Zweizeiler:

```
import xml.etree.ElementTree as ET
transactions = ET.parse("transactions/2010-01.xml").getroot().findall("transaction")
```

Ein paar Python-Versionen später wurde ElementTree in den Stand einer Standardbibliothek erhoben.«