Gute Computerbücher

Inhaltsverzeichnis

Kapitel 1

Einleitung

Hier gehe ich auf Computerbücher ein, die ich gut finde, sei es, dass sie mir beim Programmieren weitergeholfen haben, sei es, dass sie einfach schön zu lesen sind. Wie alle Auswahlen und Beurteilungen ist diese Liste natürlich subjektiv; was gut ist und was nicht, sieht jeder anders. Auch darüber, was ein "Klassiker" ist, gehen die Meinungen sicher auseinander, obwohl es ein paar Bücher gibt (wie z. B. Design Patterns oder The Art of Computer Programming), die allgemein zum Kanon der Computerliteratur gezählt werden.

Wie schon gesagt: Hier sind nur Bücher aufgezählt, die ich gut finde. Sie werden hier also keine Verrisse finden. Wenn ich ein Buch schlecht finde (immer daran denken: auch bei der Bewertung von Büchern ist vieles subjektiv), habe ich es einfach verschwiegen. Das heißt natürlich nicht, dass ein Buch, das Sie hier nicht finden, nicht gut ist. Ich habe nur eine subjektive Auswahl gelesen und besprochen, weshalb mit Sicherheit eine Menge guter Bücher fehlen.

Alle Bücher, die hier besprochen werden, habe ich gelesen (was schon ein wenig Arbeit war ;-), die Beurteilungen basieren nicht auf Durchblättern oder Hörensagen.

Die im Folgenden besprochenen Bücher sind vor allem für Programmierer gedacht. Die Themen reichen bis zum Projektmanagement, aber ohne grundlegende Programmierkenntnisse wird man mit vielen Büchern in dieser Auflistung keine rechte Freude haben. Einige Bücher verlangen sogar intensive Programmierkenntnisse, um richtig genossen werden zu können.

Der Schwerpunkt dieser Liste liegt bei Büchern in englischer Sprache. Das liegt nicht daran, dass in Deutschland keine guten Computerbücher geschrieben würden, es liegt einfach daran, dass wesentlich mehr Bücher auf Englisch als auf Deutsch erscheinen, weil der Absatzmarkt größer ist. Beachten Sie bitte, dass es von einigen dieser Bücher auch deutsche Übersetzungen gibt. Wenn Ihnen das Englische nicht so liegt, können Sie diese Bücher auch auf Deutsch lesen. Alle Anmerkungen, die ich in den Besprechungen englischsprachiger Bücher mache, beziehen sich auf das englische Original.

Meine Anmerkungen beziehen sich immer auf die Auflage, die ich selbst gelesen habe, neuere Auflagen können selbstverständlich im Inhalt abweichen: Neues kann dazugekommen sein, nicht mehr aktuelle Teile des Buches wurden eventuell gestrichen. Im Großen und Ganzen sollten die Bemerkungen über eine bestimmte Auflage auch für spätere Auflagen noch gültig sein. Es kann sein, dass einige der Bücher gar nicht mehr im Handel erhältlich sind; in diesem Fall kann eine Suche bei Zvab eventuell noch ein antiquarisches Exemplar zu Tage fördern.

Die Bücher sind nach dem Nachnamen des ersten Autors geordnet. Es gibt mehrere gleichwertige Möglichkeiten, die Bücher zu ordnen (nach Erscheinungsjahr wäre eine andere plausible Alternative). Zudem könnte man sie in Rubriken einteilen, etwa *Programmieren* oder *Projektmanagement*. Etliche Bücher wie Extreme Programming Explained oder Code Complete sprengen aber die Grenzen solch enger Rubriken und lassen sich nur mit viel Gewalt und einem virtuellen Schuhlöffel in eine Kategorie zwängen. Deswegen habe ich keine Einteilung in Rubriken vorgenommen.

Computerbücher sind ein bewegliches Ziel, schließlich werden immer wieder neue Bücher veröffentlicht. Eine aktuelle Version dieser Buchbesprechungen finden sie auf meiner Homepage (momentan auf dieser Seite, aber das kann sich ändern). Sie können sich auch gern per E-Mail direkt an mich wenden. Anmerkungen, Verbesserungsvorschläge und Kritik senden Sie bitte an *nik@drnik.org*

Kapitel 2

Danksagung

Danken möchte ich zuerst einmal den Autoren aller hier vorgestellten Bücher, denn ohne sie gäbe es all die Bücher nicht, die ich hier besprochen habe; und ohne diese Bücher hätte ich mein Buch natürlich nicht schreiben können.

Klaus Häringer danke ich fürs strenge, aber stets freundliche Korrekturlesen. Er hat mich auf etliche sprachliche Ungereimtheiten aufmerksam gemacht und stilistische Unsauberkeiten ausgeputzt. Alle verbleibenden Fehler, Ungereimtheiten sowie komische Redewendungen und Metaphern gehen auf mein Konto.

Mein Dank geht außerdem an Markus Dankesreiter, der mich auf viele der hier vorgestellten Bücher (als eins unter vielen sei stellvertretend Pragmatic Programmer) genannt) überhaupt erst aufmerksam gemacht hat.

Mein Bruder Wolfgang hat mir die Orchideenblüte, die den Umschlag dieses Buchs ziert, freigestellt (also störenden Hintergrund wie Stiele und sonstigen Krempel ausgeblendet) und mir so etliche nervenaufreibende Stunden mit dem Grafikprogramm erspart. Auch dafür ein ganz herzliches Dankeschön.

Kapitel 3

Professional XML

Richard Anderson

Mark Birbeck

Michael Kay

Steven Livingstone

Brian Loesgen

Didier Martin

Stephen Mohr

Nikola Ozu

Bruce Peat

Jonathan Pinnock

Peter Stark

Kevin Williams

Wrox

Einer der "dicken Wälzer" von Wrox, der auf über tausend Seiten so ziemlich alles abhandelt, was mit XML zu tun hat. Das geht mit einer Einführung in XML los. Neben der üblichen Einführung (was ist ein Element?), geht es hier auch darum, wie man seine Daten als XML modelliert (soll man ein bestimmtes Ding als Element oder Attribut speichern?) und ob die Daten überhaupt dazu geeignet sind, in XML modelliert zu werden. XML ist fast ein Allheilmittel, aber es gibt Daten, die in anderen Datenformaten besser aufgehoben sind. Zu einer ausführlichen Einführung in die Tiefen von XML gehören auch Namespaces, dementsprechend behandelt das Buch sie in aller Breite.

Von dort geht es weiter über DTDs und Schemas sowie DOM und SAX bis hin zu XPath und XPointer. Ein Thema ist auch der Zusammenhang zwischen XML und relationalen Datenbanken. Relationale Datenbanken sind neben Textdateien eine Möglichkeit, XML dauerhaft zu speichern. Sie sind besonders dann von Vorteil, wenn der Zugriff schnell erfolgen soll. Aber die Speicherung ist nicht ganz trivial, weshalb das Buch diesem Thema einigen Raum widmet.

Der Rest des Buchs behandelt Anwendungen von XML wie SOAP und die Konvertierung von XML in andere Formate und andere XML-Dokumente mit XSLT. Abgerundet wird das Ganze durch etliche Fallbeispiele.

Wer fast alles über XML in einem Buch zusammengefaßt haben will, und das in durchgehend guter Qualität, der liegt hier richtig. Für ganz spezielle Bedürfnisse braucht man aber weiterführende Literatur. Ein Beispiel: XSLT. Dieses Buch enthält ein einführendes Kapitel, das bei den ersten Gehversuchen mit XSLT sicher nützlich ist. Wer jedoch so gut wie alles über XSLT wissen will, kommt um XSLT von Michael Kay nicht herum.

Kapitel 4

The Java Programming Language

Ken Arnold

James Gosling

David Holmes

Addison-Wesley

Das Referenzwerk über die Programmiersprache Java, unter Beteiligung von Java-Erfinder James Gosling geschrieben. Hier wird die Syntax und Semantik von Java ausführlich beschrieben, und es werden einige der Entscheidungen, warum etwas in Java so und nicht anders geregelt ist, erläutert. Wer intensiv mit Java arbeitet, dem ist dieses Buch zu empfehlen. In Zweifelsfällen ist es die Referenz.

Zudem kann der Autor seine eigene Schöpfung gut und anschaulich erklären, was keineswegs selbstverständlich ist. So merkt man etwa Bjarne Stroustrups C++-Standardwerk *The C++ Programming Language* deutlich an, dass sich der Autor gar nicht vorstellen kann, wie man etwas an seiner wunderbaren, logischen und einfachen Programmiersprache *nicht* verstehen kann. Dem entsprechend sind seine Erklärungen haarsträubend komplizierter Sachen in C++ für den Normalsterblichen oft so gut wie nicht mehr nachvollziehbar, während sich Herr Stroustrup offenbar noch pudelwohl fühlt. Diesen Fehler machen die Autoren des hier besprochenen Buchs nicht, sie "nehmen den Leser mit" (zumindest "ein Stück weit"), wie man heute so gern sagt.

Fazit: Wer über Java in a Nutshell hinaus Einblick in die Syntax und Semantik von Java gewinnen will, sollte dieses Buch lesen. Es verdient einen Stammplatz auf dem Tisch jedes ambitionierten Javaisten.

Kapitel 5

Professional Java Server Programming

Danny Ayers

Hans Bergsten

Michael Bogovich

Jason Diamond

Matthew Ferris

Marc Fleury

Ari Halberstadt

Paul Houle

Piroz Mohseni

Andrew Patzer

Ron Phillips

Sing Li

Krishna Vedati

Mark Wilcox

Stefan Zeiger

Wrox

Ein gewaltiger Wälzer aus der Serie der gewaltigen roten Wälzer von Wrox Press. Eine ganze Reihe von Autoren schreibt hier über alle wesentlichen Aspekte der Serverprogrammierung mit Java. Das Buch beginnt mit den grundlegenden Begriffen der Serverprogrammierung wie HTTP-Request und -Response und zeigt, wie Servlets diese behandeln. Weiter geht es mit Servlets, Java Server Pages, JNDI, LDAP, JINI etc. Ausführlich gewürdigt werden auch JDBC (das Datenbank-API von Java) und Connection Pools (mit denen man Datenbankverbindungen verwalten kann, so dass man sie mehrmals benutzen kann, statt sie immer neu zu erzeugen). Ebenfalls besprochen wird JavaMail. Natürlich geht es auch um Enterprise Java Beans. Ein Teil dieser Technologien wird mit Fallbeispielen veranschaulicht.

Doch das Buch handelt nicht nur von Standardthemen wie den bereits erwähnten. Es enthält auch eine ausführliche Beschreibung von Jini und JavaSpaces, einer Technologie, die dazu dient, Geräte dezentral zu verbinden, so dass diese ihre Services anbieten und die Services anderer Geräte nutzen können. Das ist eine faszinierende Technologie, der leider nicht die Aufmerksamkeit zuteil wird, die sie verdient hätte und das Buch legt sich verdientermassen dafür ins Zeug.

Fazit: Ein einschüchternd umfassendes Werk über die Serverprogrammierung mit Java, das einen für eine ganze Weile mit dem nötigen Rüstzeug versorgt (und das einem hilft, die ganzen Buzzwords wie J2EE, UML, JNDI zu verstehen oder gegebenenfalls selbst mit diesen Begriffen um sich zu werfen). Wird die Serverprogrammierung mit Java dann zur Lieblingsbeschäftigung oder gar zum Beruf, wird man doch zu weiterführenden Büchern wie Java Server Pages oder Enterprise Java Beans greifen müssen.

Kapitel 6

Extreme Programming Explained

Kent Beck

Addison-Wesley

Extreme Programming ist ein Ansatz für Programmierprojekte, der nicht auf große Schritte setzt, sondern auf eine inkrementelle Vorgehensweise in kleinen Schritten, die dann sogleich der Realität ausgesetzt und gegebenenfalls korrigiert werden. Ein Extrembeispiel für eine Politik der großen Schritte ist das Wasserfallmodell, bei dem auf eine (eventuell jahrelange) Analyse eine umfangreiche Spezifikation folgt (diese hat oft den Umfang mehrerer Telefonbücher). Dieser haben die Programmierer dann genau zu folgen, am Schluß wird getestet und das Projekt wird an den Kunden übergeben.

Das Problem dabei ist, dass die Realität nicht mitspielt. Viele Fragen zu einem Programmierproblem sowie die Antworten darauf tauchen erst während des Projekts auf. Verfahren wie das Wasserfallmodell sind zu starr, um auf solche Änderungen zu reagieren.

Hier setzt Extreme Programming an. In kleinen Schritten (typischerweise eine Woche) werden Teilaspekte des Problems in Zusammenarbeit mit dem Kunden festgelegt. Dazu verwendet man die sogenannten *Use Cases*, kurze Prosabeschreibungen eines Problems, die auf eine Karteikarte passen. Der Kunde sollte für das Schreiben der Use Cases einen Experten für das jeweilige Fachgebiet abstellen, der mit den Programmierern zusammenarbeitet.

Dieses Teilproblem wird dann in Software umgesetzt. Passt die Struktur der Software nicht zum Problem (zum Beispiel, weil die Software zu spezialisiert ist), wird sie mit Refactoring überarbeitet. Der Gedanke dahinter ist, dass man Software nie so schreiben kann, dass sie ein Problem perfekt lösen kann. Man kann sie jedoch so schreiben, dass sie angepasst und umstrukturiert werden kann, wenn sie nicht mehr zum Problem passt. Damit beim Refactoring nichts kaputtgeht, gibt es die *Unit Tests*, die vollautomatisch die wichtigen Aspekte der Software testen. Nachdem ein Programm beim Refactoring umgebaut wurde, müssen alle Unit Tests weiterhin ohne Fehler ablaufen.

Ein weiterer wichtiger Teil des Extreme Programming ist das *Pair Programming*. Programmiert wird immer zu zweit, wobei ein Programmierer oder eine Programmiererin tippt und die oder der andere aufpasst und mitdenkt. Auf diese Weise ist die Fehlerquote deutlich geringer, als wenn ein Programmierer allein arbeitet, weil viele Flüchtigkeits- und Denkfehler vermieden werden können.

Kent Beck ist der Begründer dieser Methode und "Extreme Programming Explained" ist das grundlegende Buch zu diesem Thema. Inzwischen gibt es in der "Extreme Programming ..."-Reihe eine ganze Menge von Büchern, aber dieses Buch stellt die methodische Basis dar; es ist quasi die Bibel des Extreme Programming. Wenn man sich für Extreme Programming interessiert, sollte man es daher auf jeden Fall als erstes lesen.

Kapitel 7

XML in der Praxis

Henning Behme

Stefan Mintert

Addison-Wesley

Eine gut lesbare Einführung in XML, in einem angenehmen, humorvollen Ton geschrieben. Ich liebe kleine Gags wie: "Für die Aussprache ist schon die Schreibweise als Akronym hilfreich: *DSSSL*. Versuchen Sie das vorzulesen und es müsste zweierlei herauskommen: Erstens eine feuchtere Buchseite und zweitens so etwas wie 'Dissel'.". Im angelsächsischen Raum kommt dieser lockere Umgang mit der Sprache öfter vor (klassisches Beispiel ist Programming Perl), aber in deutschen Computerbüchern ist er leider eher selten.

Wenn Sie schon immer wissen wollten, was es mit diesem "XML" eigentlich auf sich hat, ist dies genau das richtige Buch, um sich an einem regnerischen Tag auf dem Sofa zu verkriechen und der Sache nachzuschmökern.

Ist Ihnen XML dann sympathisch geworden und Ihr Interesse geweckt, können Sie als Nächstes gleich den großen Sprung wagen und Professional XML in Angriff nehmen.

Kapitel 8

Programming Pearls

John Bentley

ACM Press

Ein Klassiker zum Thema Algorithmen und Performance. Während The Art of Computer Programming das Thema quasi "offiziell" abhandelt, ist dieses Buch eine Sammlung von Kolumnen und Artikeln. Die Betonung liegt hier auf Spaß und Kreativität, deswegen auch das "Pearls" im Titel: Hier geht es um Perlen, schöne knifflige Programmierprobleme (im Deutschen kann man sie wohl passenderweise "Kopfnüsse" nennen), bei denen das Grübeln und Lösen Spaß macht und immer eine Einsicht in einen bestimmten Aspekt des Programmierens eröffnet.

Schwerpunkte sind die fundamentalen Algorithmen wie Suchen und Sortieren. Sie dienen als Startpunkt für Exkurse über guten und sauberen Programmierstil. Damit geht das Buch in eine ähnliche Richtung wie The Practice of Programming. Ein weiteres Hauptthema ist Effizienz: Wie man die Algorithmen, wenn man sie erst einmal prinzipiell verstanden hat, so verfeinert, dass sie – nun ja – effizienter werden. Das kann bedeuten, dass sie schneller laufen oder weniger Speicherplatz verbrauchen, oder beides. Dabei betont der Autor aber stets, dass es vor allem wichtig ist, einen Algorithmus wirklich zu verstehen, bevor man an ihm herumschraubt, um ihn effizienter zu machen. Vorschnelle Optimierung ist der Tod vieler Programme.

Das Buch hat auch einen sehr beruhigenden Aspekt. Ich hatte mich jahrelang gegrämt, dass ich die vordergründig so einfache binäre Suche nicht richtig hinkriege und immer ein paar Stunden debuggen musste, bis alles stimmte. Bentley zeigt, dass ein Großteil der Programmierer (z. B. in Programmierkursen) diesen Algorithmus nicht auf Anhieb richtig hinschreiben kann, und er zitiert Knuth, laut dem die erste binäre Suche 1946 veröffentlicht wurde, die erste fehlerfreie binäre Suche dagegen erst 1962.

Wer will, kann nicht nur die Algorithmen, wie sie Bentley vorstellt, nachvollziehen, sondern sich je nach Lust und Zeit über den oft recht gehaltvollen Übungsaufgaben den Kopf zerbrechen. Aber Vorsicht: Was oft harmlos daherkommt, wird überraschend schnell knifflig, man verbeisst sich in die Knobelei und schnell ist ein ganzer Samstagnachmittag weg.

Ein lesenswertes und unterhaltsames Buch für Programmierer, die gern an kniffligen Programmierproblemen herumknabbern und einen tieferen Einblick in die vertrackte Implementierung bekannter Algorithmen gewinnen wollen.

Kapitel 9

Java Server Pages

Hans Bergsten

O'Reilly

Alles über Java Server Pages, mit vielen Beispielen garniert. Java Server Pages sind eine Technologie, die (zumindest theoretisch) Programmierung und Präsentation bei dynamischen Webseiten trennt, ähnlich wie PHP oder ASP. Verschiedene Leute mit getrennten Aufgabenbereichen wie Webdesign oder Datenbankzugriffen können gleichzeitig an der Anwendung arbeiten, ohne sich ins Gehege zu kommen.

Vor Java Server Pages gab es Servlets, Java-Programme, die Eingaben von einem Browser empfangen und als Antwort eine HTML-Seite senden. Das Problem hierbei ist, dass der HTML-Code innerhalb des Java-Codes mit Programmbefehlen erzeugt werden muss, was den Java und den HTML-Code fast untrennbar miteinander verbäckt. Hier schaffen Java Server Pages zumindest zum Teil Abhilfe, indem man den Spieß umdreht und Java in HTML einbettet. Diese Aufrufe sollen möglichst nicht zu viel Programmlogik enthalten, sondern Code außerhalb der Seite aufrufen. Zumindest in der Theorie, in der Praxis neigen viele dazu, viel explizite Programmlogik in die Seite hineinzubatzen. Damit ist gegenüber einem Servlet nicht viel gewonnen, man hat lediglich die Rollen von Java und HTML vertauscht. Das muss aber nicht sein, und der Autor zeigt in vielen Beispielen, wie man es richtig macht.

Damit man versteht, wie Servlets und JSPs arbeiten, wird zuerst einmal HTTP erklärt, die Technik, mit der Browser und Server kommunizieren. Anschließend folgt eine Einführung in Servlets. Damit man überhaupt mit Servlets und JSPs loslegen kann, muss man einen passenden Server haben; der Autor zeigt hier stellvertretend die Installation und Einrichtung von Tomcat.

Außer auf Server Pages selbst geht der Autor auch auf Taglibs ein. Während bei einer Server Page Skriptcode in eine HTML-Seite eingefügt wird, sind Taglibs Bibliotheken von Tags, die als Komponenten in eine HTML-Seite eingebaut werden können. Neben Tags wie etwa `<c:if>` für eine if-Klausel gibt es auch Tags für die Verbindung zu einer Datenbank (die Familie der `<sql:xxx>`-Tags) und vieles mehr. Das Buch zeigt auch, wie man eigene Taglibs schreibt.

Schließlich geht es um die Zusammenarbeit zwischen Java Server Pages und Java Beans und um Datenbankzugriffe. Abgerundet wird das Ganze durch eine Referenz, die alle Klassen und Taglibs beschreibt.

Kapitel 10

Perl for System Administration

David N. Blank-Edelman

O'Reilly

Dieses Buch zeigt, wie man Perl zur Systemadministration einsetzt. Ungeachtet der Diskussion über die Syntax von Perl (siehe auch die Besprechung von Programming Perl): Hier ist eines der Gebiete, in denen Perl brilliert und von vielen Administratoren rund um den Globus geschätzt wird.

Dieses Buch bietet eine solide Einführung in das Thema und ein Fülle von Beispielen, etwa zur Auswertung von Logfiles oder zum automatischen Abfangen von Spam-Mail. Und wenn Perl nicht die bevorzugte Sprache ist, kann man dieses Buch trotzdem mit Gewinn lesen. Die Beispiele sind so beschrieben, dass man die Grundidee erfassen und in eine andere Sprache übertragen kann.

Kapitel 11

Effective Java

Joshua Bloch

Addison-Wesley

Ein Java-Buch für Fortgeschrittene, mit Tips, die auch für den versierten Javaisten noch Neues bieten. Ein Großteil des Buches behandelt das objektorientierte Programmieren mit Java, mit Ratschlägen wie dem, lieber Komposition als Vererbung zu verwenden (weil Vererbung die abgeleiteten Klassen auch von der Implementation her an die Mutterklasse bindet) oder statt Konstruktoren statische Create-Methoden einzusetzen. Denn im Gegensatz zu Konstruktoren kann man von diesen beliebig viele verschiedene haben und sie darüber hinaus noch mit aussagekräftigen Namen versehen. Statt des Konstruktors

```
public Mumpitz(String s) {
...
}
```

kann man die statische Methode

```
public static Mumpitz createFromXmlData(String s) {
...
}
```

einsetzen.

Ein weiteres Thema sind C-Konstrukte und wie man diese in Java ausdrücken kann. So lautet ein Tip, Enums (die es in Java erst seit neuestem gibt), durch Klassen zu ersetzen. Für jedes Element der Enum enthält diese Klasse genau eine statische Instanz.

Fazit: Lesenswert. Wer den Einstieg in Java geschafft hat, findet hier ein Fülle von Tips, wie sie oder er noch besseres Java schreiben kann.

Kapitel 12

The Mythical Man Month

Frederick P. Brooks

Addison-Wesley

Ein Klassiker der Literatur über Projektmanagement. Die Erstausgabe des Buchs stammt aus dem Jahr 1975 und deshalb wirkt manches etwas antiquiert; so macht sich Brooks noch Gedanken über den direkten Festplattenzugriff von Programmen und über Overlays. (Na, wer erinnert sich noch an Overlays? Pascal-Veteranen heben jetzt bitte die Hand.) Viele Aussagen in diesem Buch sind aber von zeitloser Gültigkeit, so z. B. Brook's Law

> Adding manpower to a late software project makes it later

Das kann man wohl so übersetzen:

> Wenn man zu einem Softwareprojekt, das hinter dem Zeitplan hinterherhinkt, noch mehr Leute hinzuzieht, wird es garantiert noch später fertig

Eine weitere provokative These aus diesem Buch ist: "Plan to throw one away; you will, anyhow" ("Plane die erste Version deiner Software gleich zum Wegwerfen, du wirst es ohnehin tun müssen"). Die Frage ist nicht, ob man die erste Version als Prototyp baut. Das wird man laut Brooks ohnehin tun. Die Frage ist, ob man sie wegwirft und die gewonnenen Erfahrungen nutzt oder ob man sie an den Kunden ausliefert.

Dem liegt die Annahme zugrunde, dass man Softwareprojekte sowieso nicht im ersten Anlauf richtig hinbekommt, weil sie einfach zu komplex sind und man erst nach dem ersten Anlauf weiß, wie man die Sache hätte angehen müssen. Hier sieht Brooks eine Parallele zu den Ingenieurwissenschaften, bei denen man auch erst Modelle und Prototypen baut, bevor man eine ganze Fabrik, ein Wasserwerk oder eine Brücke konstruiert.

Schließlich stammt aus diesem Buch die These "No Silver Bullet"; die Behauptung, dass es in der Softwareentwicklung keine Technologie gibt, welche die Produktivität, die Zuverlässigkeit der Programme oder die Einfachheit um eine ganze Größenordnung verbessert, also mindestens verzehnfacht. Eine solche Technologie wäre das Analogon zur silbernen Pistolenkugel, die der Legende nach als letztes Mittel gegen Werwölfe eingesetzt werden kann; eine Technologie, die helfen könnte, das unbezähmbare Monster der Komplexität in der Softwareentwicklung zur Strecke zu bringen.

Brooks nimmt sich eine Reihe der Technologien vor, von denen große, bisweilen gigantische Verbesserungen erwartet werden und argumentiert dagegen. Er gesteht allen diesen Dingen zu, dass sie die Produktivität, die Zuverlässigkeit oder die Einfachheit erhöhen, zum Teil sogar erheblich, aber eben nicht um eine Größenordnung. Diese Liste enthält unter anderem

- Hochsprachen wie Ada, (heute kein so großes Thema mehr, aber damals anscheinend ein Hoffnungsträger),
- Objektorientierte Programmierung,
- Künstliche Intelligenz,
- Expertensysteme,

- Automatische Programmierung,
- Buy versus Build, also Komponente zukaufen, anstatt sie selbst zu entwickeln,

und noch etliche Dinge mehr. Brooks meint, dass alle diese Dinge keine Silberkugeln seien, weil letztlich das größte Problem nicht das Schreiben der Programme ist, sondern Spezifikation, Design und Konzeption. Und diese Arbeit kann einem bis jetzt keine Technologie abnehmen, das ist harte Arbeit, die von Menschen gemacht werden muss. Und diese Arbeit braucht einfach ihre Zeit und wird darum die Produktivität auf absehbare Zeit begrenzen.

Das Buch enthält in der neueren Auflage von 1995 auch ein Kapitel, in dem dem "No Silver Bullet" mit dem Abstand von 20 Jahren noch einmal neu bewertet wird. Zu welchen Schlüssen Brooks darin kommt, verrate ich nicht; ich will Ihnen ja nicht die ganze Spannung wegnehmen.

Kapitel 13

AntiPatterns

William H. Brown

Raphael C. Malveau

Hays W. McCormick

Thomas J. Mowbray

Wiley

Das dunkle Gegenstück zu Design Patterns: Lehrreich und mit hohem Wiedererkennungsfaktor ("Ach, so läuft das bei uns auch"). Es zeigt statt positiver Patterns, die man verwenden soll, negative Patterns, die es zu vermeiden gilt.

Ein klassisches Beispiel für ein negatives Pattern ist "Analysis Paralysis". Darunter versteht man das Phänomen, dass man bei einem Projekt immer noch eine Analyse und noch eine Analyse macht, statt mit dem Entwurf und dem Entwickeln anzufangen. Das kann mehrere Gründe haben: Es kann sein, dass man die Problemstellung, für die ein Programm entwickelt werden soll, nicht in den Griff bekommt. Es kann aber auch sein, dass politische Gründe dahinterstecken, wenn z. B. keiner der am Projekt Beteiligten als erster einen Fehler machen will, weil sein Job ohnehin schon gefährdet ist. ("Sein Stuhl wackelt", wie man so schön sagt.)

Das Buch listet dieses und andere negative Patterns auf und gibt Tips, wie man sie vermeiden kann. Es geht auch darauf ein, unter welchen besonderen Umständen man ein negatives Pattern (zeitlich begrenzt) in Kauf nehmen sollte, weil die Alternativen oder die Kur noch schlimmere Auswirkungen hätten.

Kapitel 14

An Introduction to Object-Oriented Programming

Timothy Budd

Addison-Wesley

Eine Einführung in die Konzepte objektorientierter Programmierung und wie diese Konzepte in den verschiedenen Sprachen umgesetzt werden. Folgende Sprachen sind mit von der Partie: Object Pascal, Smalltalk, Objective-C, C++ und Java. Diese Tour durch das objektorientierte Programmieren in verschiedenen Sprachen ist sehr erhellend. Man sieht, wie die verschiedenen Aspekte der OO-Programmierung wie Vererbung, Abstraktion, Polymorphismus und Kapselung in den verschiedenen Sprachen umgesetzt werden. Das Ganze wird mit etlichen Fallbeispielen untermauert, deren Implementierung in all diesen Sprachen gezeigt wird. Diese Beispiele sind durchaus gehaltvoll; so wird etwa das Eight-Queens-Problem durchexerziert, bei dem es darum geht, acht Damen so auf einem Schachbrett anzuordnen, dass keine die andere schlagen kann.

Wenn man mit einer objektorientierten Programmiersprache anfängt, sollte man dieses Buch auf jeden Fall lesen, um das nötige Hintergrundwissen und einen Überblick zu bekommen. Dieses Buch hat allein schon deswegen bei mir einen Stein im Brett, weil Objective-C (eine Sprache, bei der ein Smalltalk-ähnliches Objektsystem auf C aufgesetzt wird und der leider nie der ganz große Durchbruch gelang) darin vorkommt.

Kapitel 15

Pragmatic Project Automation

Mike Clark

Dieses Buch ist Teil des Pragmatic Starter Kits der Pragmatic Programmer Andrew Hunt und David Thomas. Ihr Ziel ist, wie der Name schon nahelegt, eine pragmatische Herangehensweise an Programmierprojekte. Nachdem sie ihre Ansichten in Pragmatic Programmer dargelegt haben, legen sie nun zur praktischen Umsetzung das Pragmatic Starter Kit nach, eine Buchreihe, die einzelne Aspekte der Programmentwicklung im Detail behandelt, wobei sie meist nicht als Autoren, sondern nur als Herausgeber auftreten. Während die beiden ersten Bände des Starter Kit sich mit Versionskontrollsystemen wie CVS und mit Unit Tests beschäftigen, handelt dieser Band von der Automatisierung.

Das bedeutet zunächst, den Prozess zu automatisieren, der die Software "versandreif verpackt" für den Kunden erstellt. Hier kommen Tools wie Ant und Make, aber auch Shell-Skripte zum Einsatz. Die Automatisierung geht aber darüber hinaus, indem die Test automatisiert und regelmäßig ausgeführt werden, entweder zu bestimmten Zeitpunkten oder bei bestimmten Ereignissen wie etwa beim Einchecken geänderten Codes ins Versionskontrollsystem. Das Buch geht aber noch weiter, indem es zeigt, wie man bei fehlgeschlagenen Tests oder sonstigen Problemen eine Nachricht verschickt, etwa per E-Mail oder per SMS.

Ein etwas skurriles Beispiel zeigt schließlich, wie man die Aufmerksamkeit der Entwickler auf Probleme lenkt, indem man ungewöhnliche Gerätschaften einsetzt, nämlich eine grüne und eine rote Lavalampe. Ist alles im sprichwörtlichen grünen Bereich, blubbert die grüne Lavalampe beruhigend vor sich hin. Gibt es dagegen Probleme, alarmiert die rote Lavalampe alle Beteiligten.

Das Buch ist im für diese Reihe typischen lockeren Ton geschrieben. So stellt etwa "Joe the Programmer" immer wieder Zwischenfragen, zum Beispiel, warum man nicht diese oder jene Software für einen bestimmten Zweck verwendet oder warum man in einem bestimmten Fall nicht so und so vorgeht. Aufgrund seiner Kürze ist es zudem schnell durchgelesen. Für alle, die gern skripten, ist es eine unterhaltsame Lektüre und ein Denkanstoß. Für Projektleiter sollte es Anlass sein, jemanden, der gern skriptet, damit zu beauftragen, zumindest einige der Ideen aus diesem Buch umzusetzen.

Kapitel 16

Program Generators with XML and Java

J. Craig Cleveland

Prentice Hall

Die "Charles F. Goldfarb Series on Open Information Management" besteht aus Büchern zum Thema XML im weitesten Sinn. Dieses Buch handelt davon, wie man mit Programmen Programme erzeugt, statt diese von Hand zu schreiben. Ein Programm, das dazu da ist, andere Programme zu erzeugen, nennt man einen *Program Generator*. Gründe dafür, beim Entwickeln so vorzugehen, gibt es viele. Ein Grund, der in diesem Buch besonders hervorgehoben wird, ist, dass alle Programme zu einer *Domäne* gehören, also einem bestimmten, eingrenzbaren Problembereich. Typisch für solche Domänen ist, dass man Programme braucht, die in vielen Teilen gleich sind, in anderen aber an die variablen Aspekte der Domäne angepasst werden müssen. In diesem Fall ist es sinnvoll, diese Programme automatisch zu erzeugen und die variablen Bestandteile über den Programmgenerator zu konfigurieren.

Dieses Buch zeigt ausführlich die Techniken, die beim Schreiben von Programmgeneratoren angewendet werden. Es zeigt etliche verblüffende Beispiele, so etwa eine Webseite, auf der sich der Anwender ein Applet, das ein Balkendiagramm anzeigt, selbst konfigurieren und dann automatisiert herstellen kann. Obwohl die gezeigten Techniken allgemeingültig sind, legt der Autor den Schwerpunkt auf XML als Beschreibungssprache für die zu generierenden Programme.

Fazit: Auch wenn man die gezeigten Techniken nie einsetzen wird, ist das Buch trotzdem lesenswert, weil es den Horizont erweitert und zeigt, was bei der Softwareentwicklung alles möglich ist. Für alle, die den Einstieg in generative Programmierung finden wollen und die so an der Hand genommen werden wollen, dass sie bald eigene Programmgeneratoren schreiben können, ist dies das richtige Buch. Wer tiefer einsteigen und mehr über Hintergründe und Theorie der generativen Programmierung erfahren will, sollte (zusätzlich) zu Generative Programming greifen.

Kapitel 17

Generative Programming

Krzysztof Czarnecki

Ulrich W. Eisenecker

Ein Buch über generative Programmierung, also darüber, wie man Programme schreibt, die Programme schreiben. Es geht noch tiefer als Program Generators with XML and Java, ist allerdings auch deutlich trockener und sperriger zu lesen. Neben viel Code und etlichen Beispielen gehen die Autoren auch umfassend auf die Theorie hinter der generativen Programmierung ein.

Die in diesem Buch gezeigten Programme sind von gehobenem Niveau. Verblüffend ist die Idee, den Präprozessor von C/C++ (genau, der mit den `#ifdefs` und den `#defines`) als eigene, Lisp-artige funktionale Programmiersprache einzusetzen. Manch einem, der beim Wort C++ schon Unbehagen verspürt, mögen sich bei dieser Idee vollends die Zehennägel aufrollen. Auf der anderen Seite können es gerade solche Ideen sein, die auch altgediente Programmierer noch dazu bewegen, die eingetretenen Pfade zu verlassen und auch mal über den Tellerrand zu gucken. (Eigentlich ein ziemlich schiefes Bild, wo gibt es schon Pfade, die am Rand mit Tellern gesäumt sind. Aber ich wollte die beiden Ausdrücke unbedingt in diesem Buch unterbringen, also warum nicht hier?)

Wen die Hintergründe der generativen Programmierung interessieren, der sollte dieses Buch unbedingt lesen. Wer nur mal wissen will, wie sich generative Programmierung "anfühlt" und ein paar Codebeispiele selbst ausprobieren will, der ist mit Program Generators with XML and Java besser bedient.

Kapitel 18

Thinking in Java

Bruce Eckel

Prentice-Hall

Das "The Definitive Introduction..." im Untertitel erscheint auf den ersten Blick etwas hochgegriffen, kommt aber der Sache ziemlich nahe. Ein umfassendes Werk über Java für den fortgeschrittenen Programmierer. Dieses Buch macht da weiter, wo die Einsteigerbücher aufhören und beantwortet die Fragen, die sich jedem stellen, der intensiver mit Java zu tun hat.

Am Anfang steht eine intensive Einführung in das objektorientierte Programmieren mit Java, die auch gehaltvolle Themen behandelt, etwa, wie man im Detail mit der `finalize()`-Methode arbeitet, was die Unterschiede zwischen den vier Sorten von inneren Klassen sind, wann man welche Sorte einsetzt und wann man Komposition und wann man Vererbung verwenden sollte. All das findet nicht im luftleeren Raum statt, sondern wird mit vielen Beispielen begründet.

Weitere Themen sind Collections (Klassen zum Aufbewahren und Verwalten von Objekten wie Vektoren, Hashtables oder Sets), Fehlerbehandlung, Introspection (das Abfragen der Eigenschaften eines Objekts zur Laufzeit), Threads und das Java Native Interface (JNI) das Java- mit C-Code verbindet.

Fazit: Ein sehr empfehlenswertes Buch für den fortgeschrittenen (Java-)Programmierer.

Kapitel 19

XML Hacks

Michael Fitzgerald

O'Reilly

100 Tips und Tricks zum Umgang mit XML. Die Bandbreite reicht von grundlegend bis extrem fortgeschritten. Das Buch beginnt mit einer kurzen Beschreibung der Syntax von XML, mit Entities und ähnlichem. Wer ein XML-Hacker ist, wird diese Grundlagen wahrscheinlich schon mitbringen.

Dann wird es zunehmend gehaltvoller. Einige Highlights: Hack #12, der den sehr guten Nxml-Mode für Emacs empfiehlt. Hack #58, der zeigt, wie man mit Extensions für XSLT (EXSLT) Dinge tun kann, die man in XSLT alleine nicht zustande bringt. Hack #49, der die Tag-Soup-Version von Saxon vorstellt, die wohl kaum jemand kennen dürfte. (Mit Tag Soup ist "schludriges" (X)HTML gemeint, das kein wohlgeformtes XML ist und das deswegen von einem reinen XML-Parser nicht gelesen werden kann.)

Ein ganzes Kapitel ist XML-Schema und DTDs gewidmet, wie man sie nutzt und wie man sie ineinander umwandelt. Außerdem wird demonstriert, wie man bereits existierende Dokumente als Vorlage nutzen kann, um Schemas oder DTDs zu erzeugen. Diese Dinge findet man in kaum einem anderen XML-Buch.

Einige wenige Hacks sind ein bisschen unpraktisch, z. B. Hack #52, bei dem in einer Tabelle mit Hilfe eines XSL-Stylesheets aufsummiert wird. Wer wird sich die Mühe machen, ein dermaßen umfangreiches, handgemachtes Stylesheet zu schreiben, das nur für genau eine Sorte von Dokument geeignet ist? Wenn das Beispiel nur als Anregung für ein allgemeineres Stylesheet gedacht ist, sollte das der Autor auch explizit sagen. Ähnlich ist es in Hack #40, wo zwei Stylesheets gezeigt werden, die nur auf genau ein Dokument passen. Die Idee, den Attribut-vs.-Element-Disput zu entschärfen, ist gut, aber kaum jemand wird dafür den Aufwand treiben, für jedes seiner Dokumente zwei spezifische, kaum wiederverwendbare Stylesheets zu schreiben.

Fazit: Bis auf ein paar kleinere Ausnahmen sind die Hacks ziemlich gehaltvoll und fundiert. Das Buch hat ein paar kleinere Fehler (z. B. die drolligen Eindeutschungen in Hack #53 oder die mißverständliche Bemerkung über die Rolle des Kommas in CSV-Dateien in Hack #68), aber die bringen einen einigermaßen mit XML Vertrauten nicht aus dem Takt. Auch XML-Profis können bei diesem Buch noch etwas dazulernen. Es ist das "XML-Buch, in das man schaut, wenn man in den anderen XML-Büchern keine Antwort gefunden hat". Das gilt natürlich auch für das Internet, denn im Internet gibt es fast alles, man muss es nur finden. Sagen Sie jetzt nicht "Google". Google hilft oft beim Suchen, aber nur dann, wenn einem die richtigen Stichwörter für die Suche einfallen. Hat man ein Buch, werden einem die Probleme serviert und oft fällt einem dann erst auf, dass es genau die Dinge sind, über die man selbst schon gegrübelt hat oder an denen man beinahe verzweifelt ist.

Kapitel 20

Java in a Nutshell

David Flanagan

O'Reilly

Die "Java in Nutshell"-Reihe von David Flanagan umfasst neben dem grundlegenden "Java in a Nutshell" noch weitere drei Bände: "Java Examples in a Nutshell", "Java Foundation Classes in a Nutshell" und "Java Enterprise in a Nutshell".

Alle Bücher dieser Reihe sind knapp, trocken und sehr fundiert geschrieben. Vom Anspruch her sind sie auf erfahrene Programmierer ausgelegt, Einsteiger sollten zu anderen Büchern über Java greifen.

Das namensgebende Buch dieser Reihe bietet eine Einführung in Java. Es behandelt unter anderem Syntax, Datenstrukturen und objektorientiertes Programmieren. Auch kompliziertere Aspekte wie die vier verschiedenen Sorten von "inneren Klassen", die es in Java inzwischen gibt, werden beschrieben. Der Rest des Buchs (ungefähr die Hälfte) besteht aus einer Referenz der verschiedenen Packages und der in ihnen enthaltenen Klassen. Im ersten Band werden die grundlegenden Packages (`java.lang`, `java.io` etc.) beschrieben, speziellere Packages (wie etwa `javax.servlet`) werden in den folgenden Bänden behandelt.

Nun kann man sich darüber streiten, ob es sinnvoll ist, große Teile eines Buches mit Klassendokumentation zu füllen. Schließlich gibt es das alles auch im Internet, dazu ist die Dokumentation dort immer auf dem neuesten Stand. Ich persönlich finde es sehr angenehm, Sprachbeschreibung und Klassenreferenz in einem Band zu haben. So hat man alles auf einen Blick in komprimierter Form, auch wenn man mal nicht mit dem Netz der Netze verbunden ist. Und so schnell ändert sich die Beschreibung der Klassen nicht, viele Klassen haben sich über Jahre kaum verändert.

Fazit: Für Programmierer, die eine fundierte Beschreibung der wesentlichen Aspekte von Java suchen, sind diese Bücher (auch angesichts des moderaten Preises) gut geeignet. Auf dem Schreibtisch eines Java-Programmierers sollte der erste Band immer in Griffweite sein. Für Spezialgebiete oder eine tiefere Analyse der Sprache sollte man auf weiterführende Bücher wie etwa The Java Programming Language, Effective Java oder Thinking in Java zurückgreifen.

Kapitel 21

Refactoring

Martin Fowler

Addison Wesley

Mit Refactoring bezeichnet man das "Ausputzen" von Code. Wenn ein Programm geschrieben ist und (so halbwegs) tut, was es soll, ist das noch lang kein Grund, sich zurückzulehnen und es nicht mehr zu verbessern. Man sollte permanent versuchen, die Struktur des Programmes klarer, modularer und übersichtlicher zu gestalten. Einerseits, um die Wartung und Weiterentwicklung des Programms zu erleichtern, andererseits einfach, um es besser zu machen. Refactoring ist eine der Säulen des Extreme Programming, über das Sie z. B. in Extreme Programming Explained mehr erfahren können.

Martin Fowler zeigt, wie man Code erkennt, der dringend der Überarbeitung bedarf; er nennt das "Bad Smells in Code", also (virtuelle) üble Gerüche im Programmcode. Gute Programmierer haben ein (ebenfalls virtuelles) Näschen für solche Gerüche. Fowler gibt eine Anleitung, wie man beim Refactoring vorgehen kann; diese Anleitungen sind in einer Reihe von kleineren Regeln wie etwa "Extrahiere eine gemeinsame Superklasse" formuliert. Um das Ganze anschaulich zu machen, sind diese Anleitungen mit vielen kleinen Codebeispielen garniert.

Das Refactoring sollte man natürlich nicht einfach mit blindem Eifer in Angriff nehmen, denn die Gefahr ist groß, dass man etwas kaputtmacht. Darum ist das Refactoring untrennbar mit Unit Tests verbunden, die sicherstellen, dass der Code vor und nach dem Refactoring tut, was er soll.

Kapitel 22

The Ruby Way

Hal Fulton

Sams Publishing

Wer Programming Ruby schon gelesen hat, Gefallen an Ruby gefunden hat und tiefer in die Materie einsteigen will, der sollte dieses Buch lesen. Es behandelt weiterführende Themen und bietet unter anderem einen tieferen Einblick in das Objektmodell von Ruby. Ein weiteres Thema ist die Verwendung von Ruby zur Systemadministration. Auch für Ruby typische Idiome werden gezeigt, also Ausdrücke, die für Ruby charakteristisch sind und die den unverwechselbaren Charme dieser Sprache ausmachen. Nicht zu kurz kommen Threads, die in Ruby genau wie in Java als Objekte implementiert sind. Zudem beschreibt Fulton die Philosophie, die hinter Ruby steckt und gibt Tips für Programmierer, die von Perl bzw. Python auf Ruby umsteigen wollen.

Kapitel 23

Design Patterns

Erich Gamma

Richard Helm

Ralph Johnson

John Vlissides

Addison-Wesley

Dieses Buch ist ein Klassiker, den wirklich jeder Programmierer gelesen haben sollte. Ein Pattern ist ein Muster, das in Software immer wieder vorkommt und das man abstrahieren und in eine eigene Klasse (oder einen Satz von Klassen) extrahieren kann. Abgesehen vom praktischen Nutzen des Extrahierens (man hat dann allgemeingültige Klassen, die das Pattern implementieren und die man immer wieder anwenden kann) hilft das Extrahieren von Patterns in eigene Klassen auch dabei, die Muster leichter zu erkennen und zu verstehen. Man bekommt mit der Zeit sozusagen einen Jagdinstinkt für Patterns.

Ein populäres Beispiel ist ein "Iterator", mit dem man über Elemente beliebiger Datenstrukturen iteriert, um diese Elemente zu bearbeiten. Bedingung dafür ist natürlich, dass das, was man mit den Elementen tun will, nicht von der Datenstruktur abhängt, in der sie stecken.

Ein Beispiel für einen Iterator ist:

```
class Iterator {
boolean hasMore();
Object next();
}
```

Der Punkt ist, dass es egal ist, um welche Datenstruktur es sich handelt (beispielsweise einen Vektor oder eine verknüpfte Liste oder gar einen Baum oder Graph); man hat immer die gleiche Art von Iterator und muss sich nicht um die Datenstruktur selbst kümmern, wenn man nur an den in ihr enthaltenen Elementen interessiert ist.

Es gibt eine Menge solcher Muster und seitdem das Design-Patterns-Buch erschienen ist, sind eine Menge hinzugekommen. Das Buch listet einen Grundvorrat an Patterns auf und gibt auch Hinweise, wie man diese selbst finden kann.

Kapitel 24

C Interfaces and Implementations

David R. Hanson

Addison Wesley

Wer denkt: "Ach, sauber programmieren kann man sowieso nur mit objektorientierten Sprachen und C ist halt was für echte Hacker, ein C-Programm *muss* unübersichtlich sein", der irrt. Es gibt auch in C eine Technik, um saubere Programme zu schreiben: ADTs (Abstract Data Types). Das bedeutet, ein Ding (analog zu einem Objekt in objektorientierten Sprachen), z. B. einen Vektor, einen String, oder einen Baum so zu verpacken, dass als Schnittstelle nur ein Satz von Funktionen verwendet wird. Die Implementation hingegen bleibt unsichtbar (Natürlich kann man sie nachschauen, wenn man den Quellcode hat, aber ein braver Programmierer tut das nicht).

Auf diese Weise wird vermieden, dass die anderen Teile eines Programms Kenntnis von den "Innereien" einer Library bekommen und die einzelnen Teile des Programms untrennbar miteinander verbacken werden. Stattdessen bekommt man modulare Programme, deren einzelne Teile nicht zu eng gekoppelt sind.

Ein minimales Beispiel für einen solchen ADT bietet das folgende Interface für einen Vektor.

```
void Vector_add(Vector* v, void* elem);
void* Vector_get(Vector* v, int index);
void Vector_set(Vector* v, int index, void* elem);
.
.
.
```

Das sieht doch schon nach einem Schritt in Richtung Objektorientierung aus: Statt direkt in den Eingeweiden eines Array herumzuwühlen, mit all den damit verbundenen Unappetitlichkeiten und Risiken wie etwa Speicherverwaltung und illegale Zugriffe jenseits der Arraygrenzen, hat man eine sauber definierte Schnittstelle, alle Unappetitlichkeiten finden jenseits der Schnittstelle statt und unsere Hände bleiben sauber.

Natürlich ist das keine komplette objektorientierte Programmierung. Was fehlt, sind Vererbung und Polymorphismus, also die Fähigkeit, eine Methode mit demselben Namen je nach dem genauen Typ des Objekts, zu dem sie gehört, verschieden zu implementieren. Das in unzähligen OO-Kursen und -Büchern strapazierte Beispiel dafür ist eine Klasse *Shape*, deren Unterklassen *Rectangle*, *Triangle* und *Circle* eine Methode *draw* verschieden implementieren, und zwar logischerweise so, dass das entsprechende Objekt gezeichnet wird. Das können ADTs nicht bieten. Trotzdem sind sie ein gewaltiger Schritt in Richtung sauberer und robuster Programme.

Mir hat das Buch auch deswegen gut gefallen, weil ich an C++ schon oft verzweifelt bin (es ist mir einfach zu unübersichtlich und komplex) und dieses Buch einen Weg zeigt, auch in C sauber zu programmieren und größere Projekte übersichtlich und modular zu gestalten.

Kapitel 25

Compiler Design in C

Allen I. Hollub

Prentice-Hall

Hier wird gezeigt, wie man einen C-Compiler baut, auf fast 900 Seiten und in einer erschlagenden Detailfülle. Das Buch beginnt mit einer Einführung in Grammatiken und zeigt dann in ausführlichen Codebeispielen, wie ein Compiler aufgebaut ist. Der Compiler wird komplett "zu Fuß" aufgebaut. Das führt einerseits zu viel Code, der für manchen vielleicht zu ermüdend zu lesen ist. Auf der anderen Seite lernt man aber wirklich jedes Schräubchen eines Compilers kennen, und zwar mit Vor- und Nachnamen.

Neben berufsmäßigen Compilerbauern ist das Buch für alle interessant, die wissen wollen, wie ein Compiler genau funktioniert und wie die Anweisungen einer Hochsprache wie C zu Anweisungen werden, die ein Computer versteht. Außerdem ist es für alle von Interesse, die ein wenig Respekt vor Tools wie Lex und Yacc gewinnen wollen, denn wer dieses Buch gelesen hat, weiß, wieviel Fieselarbeit diese Tools uns eigentlich abnehmen.

Kapitel 26

The Pragmatic Programmer

Andrew Hunt

David Thomas

Addison Wesley

Eine Sammlung wertvoller Tips und Richtlinien für Programmierer. Sollte man gelesen haben, da es voll von Denkanstössen und unmittelbar in der Praxis umsetzbaren Ratschlägen ist; zudem erleichtert es die Arbeit, wenn man zumindest ein paar der vorgestellten Grundsätze beherzigt.

Die oberste Maxime der beiden Autoren und der wichtigste Rat an den Leser ist: immer pragmatisch an die Sache herangehen, also nach der einfachsten Lösung suchen, die den Zweck erfüllt. Unter diesem Motto stellen sie verschiedene Prinzipien vor, wie etwa DRY (Don't repeat yourself), das heißt, alles in einem Programm nur einmal zu sagen und Code nicht durch Kopieren und Einfügen zu vervielfältigen. Die Strafe fürs Kopieren und Einfügen ist ein gewaltiger Wartungsaufwand, wenn das Programm geändert werden muss und viele nahezu (aber eben nicht ganz, weil man die Schnipsel nach dem Einfügen natürlich "ein wenig angepasst" hat) gleichartige Stellen im Programmcode aufgespürt und überarbeitet werden müssen.

Weitere Prinzipien sind: Verwende für möglichst vieles in einem Programm, wie Datenformate oder Kommunikationsprotokolle, einfachen Text statt Binärformaten. Dann kann man leichter feststellen, wo Fehler liegen, und die Daten mit jedem Editor überprüfen und gegebenenfalls manipulieren. Bei einem Binärformat kann der Mensch dagegen nur schwer mitlesen, entsprechend schwer tut er sich beim Verstehen und Korrigieren von Fehlern. Großen Wert legen Hunt und Thomas auch auf Tests, testen kann man nach ihrer Meinung gar nicht genug, und schließlich auf Automatisierung. Was auch immer automatisch ablaufen kann, sollte man automatisch ablaufen lassen. Zu diesem Thema ist in der Serie *Pragmatic Starter Kit* ein eigenes Buch erschienen, und zwar Pragmatic Project Automation.

Wichtig ist auch, sich für sein Handwerk einen Satz an zuverlässigen Werkzeugen zuzulegen und danach zu streben, diese wirklich gut zu beherrschen (nicht umsonst ziert den Umschlag des Buchs das Bild eines Zimmermannshobels). Dazu gehört auch, sich für einen Texteditor zu entscheiden. Sei das nun Vi oder Emacs oder ein anderer Editor, Hauptsache es ist einer, der sich über die Tastatur steuern lässt, mit der Profis immer noch deutlich schneller sind als mit der Maus und einer grafischen Benutzeroberfläche. Diesen Editor sollte man in- und auswendig kennen und im Schlaf beherrschen, so wie ein Zimmermann sein Werkzeug.

Ein Buch, das in die gleiche Kerbe schlägt, ist The Practice of Programming, in dem Simplicity, also Einfachheit, eine der Maximen ist. The Pragmatic Programmer ist lockerer geschrieben und leichter zu lesen, The Practice of Programming enthält dafür mehr zum Selber-Knobeln und Nachprogrammieren. Wenn Sie die Zeit haben, lesen Sie am besten beide Bücher.

Kapitel 27

The Tao of Programming/The Zen of Programming

Geoffrey James

InfoBooks

Zwei dünne Büchlein, die ich in einem Eintrag zusammengefasst habe. Beide enthalten tiefe Weisheiten übers Programmieren, die mit einem Augenzwinkern serviert werden.

Eine Kostprobe aus dem *Tao of Programming*:

"The Tao gave birth to machine language. Machine language gave birth to the assembler.

The assembler gave birth to the compiler. Now there are ten thousand languages.

Each language has its purpose, however humble. Each language expresses the yin and yang of software. Each language has its place within the Tao.

But do not program in Cobol if you can avoid it."

("Das Tao gebar die Maschinensprache. Die Maschinensprache gebar den Assembler.

Der Assembler gebar den Compiler. Nun gibt es zehntausend Sprachen.

Jede Sprache hat ihre Bestimmung, wie bescheiden diese auch sein mag. Jede Sprache drückt das Yin und Yang von Software aus. Jede Sprache hat ihren Platz innerhalb des Tao.

Aber programmiere nicht in Cobol, wenn du es vermeiden kannst.")

Ähnlich wie *The Tao of Programming* ist *The Zen of Programming* vom selben Autor, nur diesmal eben mit Zen, also Koans etc. Ein schräges Lesevergnügen.

Kapitel 28

eBay Hacks

David A. Karp

O'Reilly

Hundert Tips und Tricks zum Umgang mit Ebay. Das fängt an beim Bieten: Welche Bietstrategien gibt es? Lohnt sich der Einsatz eines Snipers (Programme oder Internet-Dienste, die automatisch im letztmöglichen Moment ein Gebot abgeben)? Wie steht es mit den moralischen Aspekten beim Einsatz eines Snipers? (Menschliche Bieter haben gegen Sniper wenig Chancen, was die Bieterei unfair macht.)

Weiter geht es zum Verkaufen. Hier geht es darum wie man seine Artikel optimal präsentiert. Das bedeutet unter anderem, dass man die Beschreibung in der Titelzeile, die in der Liste der Auktionen sichtbar ist, und im Text so formuliert, dass der Artikel bei einer Suche möglichst gut gefunden wird.

Ein eigenes Kapitel ist dem Thema gewidmet, wie man seine Artikel so fotografiert, dass sie möglichst attraktiv aussehen. Der Autor empfiehlt rät aber davon ab, Bilder zu schönen, um Mängel und Gebrauchsspuren zu kaschieren. Das führt nur zu enttäuschten Bietern, zu Ärger und evtl. zu schlechten Bewertungen (und generell zu einem schlechten Ebay-Karma).

Der Rest des Buchs behandelt Themen für Fortgeschrittene. Das Buch macht seinem Titel alle Ehre, hier wird richtig "gehackt", im besten Sinn des Wortes. In vielen Hacks werden Skripte in Perl und Javascript gezeigt. Man lernt, wie man automatisch eine E-Mail an den Gewinner der Auktion schickt, wie man per Skript unzuverlässige Bieter findet, wie man seine Auktionen per Skript im Bündel einstellt und vieles mehr.

Wer ein Geschäft gründen will oder ein solches schon hat, um das Handeln bei Ebay als Lebensunterhalt (oder zumindest als Zugewinn) zu betreiben, bekommt Tips, wie man es richtig macht.

Die letzten 20 Hacks sind schließlich für sehr fortgeschrittene Ebayer; hier geht es um die Ebay-API, eine Programmierschnittstelle, die Ebay (kostenpflichtig) zur Verfügung stellt. Damit ist man nicht mehr darauf angewiesen, die HTML-Seiten per Skript vom Bildschirm zu "scrapen", um die Funktionen von Ebay automatisiert zu nutzen (was das Problem birgt, dass die Skripte schon bei kleinen Änderungen im Layout der Seiten nicht mehr funktionieren). Stattdessen kann man eine XML-Schnittstelle einsetzen, die von Ebay dokumentiert ist.

Diese Besprechung bezieht sich auf die (sehr gute und flüssige) deutsche Übersetzung von Kathrin Lichtenberg. Für deutsche Ebayer ist es auf jeden Fall sinnvoll, diese Version zu lesen. An vielen Stellen hat die Übersetzerin das amerikanische Original an die deutschen Verhältnisse angepasst. Etliches, was beim amerikanischen Ebay (ebay.com) funktioniert, geht in Deutschland (ebay.de) nicht, und umgekehrt. Die Abschnitte aus dem Original, die sich auf das amerikanische Ebay beziehen, enthält das Buch aber auch, sie sind mit einem Fähnchen gekennzeichnet, um darauf hinzuweisen, dass sie sich nur auf ebay.com anwenden lassen.

Kapitel 29

XSLT

Michael Kay

Wrox

2nd Edition

Ein extrem umfangreiches Werk über XSL-Stylesheets. Es ist meiner Meinung nach die Referenz zu diesem Thema. Dieses Buch listet nicht nur die Elemente und Funktionen auf, sondern es erklärt auch ausführlich die Funktionsweise von XSL (das, nebenbei bemerkt, auch bei der Herstellung dieses Buchs und seiner Umsetzung für die Online-Präsentation beteiligt ist). Es erklärt den theoretischen Unterbau von XSL; alle beteiligten Dokumente, Input, Output und das Stylesheet selbst werden als Bäume repräsentiert.

Extrem genau geht der Autor auf Expressions und ihre Anwendung in XPath ein, sie werden in ihre kleinsten Fitzelchen zerlegt und jedes Fitzelchen wird beschrieben und katalogisiert. Wollen Sie XSL hauptsächlich anwenden, können Sie diese Kapitel (speziell das über Expressions) überfliegen. Die können Sie immer noch lesen, wenn Sie zufällig auf einer Berghütte eingeschneit sein sollten, mit nichts als Dosenfleisch, Tee, Brot und natürlich dem Buch. (Wer ohne XSLT-Buch in die Berge geht, ist natürlich selbst schuld, wenn ihm dann langweilig wird.)

Nach all den Grundlagen (die schon mehr Seiten umfassen als die meisten anderen Computerbücher) zeigt der Autor dann an ausführlichen Beispielen die verschiedenen Typen von Stylesheets (Push, Pull, Computational). Er präsentiert eine Reihe von XSLT-Prozessoren, wobei er seinen eigenen, Saxon, nicht einseitig bevorzugt, sondern allen gebührende Aufmerksamkeit zukommen lässt.

An Umfang und Tiefgang ist dieses Buch dem XSL-Buch von Doug Tidwell (bei O'Reilly erschienen) deutlich überlegen. Wem allerdings das Buch von Michael Kay gar zu wuchtig erscheint, der kann erstmal mit dem Buch von Doug Tidwell anfangen.

Kapitel 30

The Practice of Programming

Brian W. Kernighan

Rob Pike

Addison Wesley

Eine Sammlung von "Best Practices" für den Programmiereralltag, ähnlich wie der Pragmatic Programmer. Die Grundsätze guter Programmierung sind bereits auf dem vorderen Buchumschlag zusammengefaßt: Simplicity, Clarity, Generality, also Einfachheit, Klarheit, Allgemeingültigkeit. Um diese Ziele zu erreichen, geben die Autoren Tips aus ihrer langjährigen Praxis.

Ein Grundsatz ist, ausführlich zu testen, und zwar schon während man programmiert. Während Funktionstests sicherstellen sollen, dass man korrekten Code schreibt, stellen Performance-Tests sicher, dass Programme die Anforderungen der Anwender an ihre Leistungsfähigkeit (beispielsweise die Anzahl der Anfragen, die ein Webserver pro Sekunde beantworten kann) erfüllen. Für Performance-Tests setzt man gern Profiler ein, also Programme, die ein anderes Programm während der Laufzeit überwachen und analysieren, wieviel Zeit diese in ihren jeweiligen Routinen verbringen. Auf diese Weise kann man die kritischen Stellen finden, die in einem Programm die meiste Rechenzeit kosten und diese gezielt optimieren.

Ein weiteres Thema ist Notation. Im Prinzip kann man mit jeder Programmiersprache alles machen. Für den Menschen aber ist die Notation wichtig, also die Art und Weise, wie ein Programm formuliert wird. Wenn diese gut zum Problem passt, ist es leichter, dieses Problem mit Hilfe einer Programmiersprache in den Griff zu bekommen. So mögen vielen die Regulären Ausdrücke, wie man sie in Perl, Sed, Awk oder Grep verwendet, kryptisch erscheinen. (Reguläre Ausdrüe beschreiben eine Klasse von Zeichenketten; so beschreibt etwa `[AB]lala.*` alle Zeichenketten, die mit `A` oder `B` beginnen, gefolgt von der Zeichenkette `lala.` und einer weiteren beliebigen Zeichenkette.) Was auf den ersten Blick kryptisch erscheint, ist nach einer Weile ein Segen, denn ohne diese kurze Notation könnte man das Ganze nur in einem viel längeren, unübersichtlichen Ausdruck beschreiben. Übungsaufgabe: Versuchen Sie, eine Notation zur Beschreibung von Zeichenketten zu erfinden, die ausführlicher ist als Regular Expressions (so könnte man den Anfang des vorigen Ausdrucks etwa als `if(class('A', 'B'))` formulieren). Dabei soll diese Notation aber genauso gut auf einen Blick zu erfassen sein wie Regular Expressions. Viel Glück!

Notation spielt also eine wesentliche Rolle dabei, wie der Mensch einen Sachverhalt erfassen kann. Die Autoren plädieren dafür, wo immer möglich eine Sprache auszuwählen, deren Notation zum jeweiligen Zweck passt. Um ihr Anliegen zu verdeutlichen, zeigen sie ein Programm in den Sprachen C, C++, Java, Awk und Perl, das Pseudozufallstexte erzeugt. Sie vergleichen, wie sich das Problem in der jeweiligen Sprache beschreiben und lösen lässt. Darüber hinaus vergleichen sie die Performance der Programme.

Ein weiteres Thema des Buchs ist Automation. Als Beispiel dient ein Skript, das aus einem C-Header, der Zahlenwerte für Fehler und eine Beschreibung des Fehlers in einem Kommentar enthält, eine Liste von Fehlermeldungen erstellt, mit der ein Programm die zu einer Fehlernummer passende Meldung ausgeben kann. Dies ist ein Beispiel für generative Programmierung, dem Hauptthema der Bücher Generative Programming und Program Generators with XML and Java.

Fazit: Ein sehr empfehlenswertes Buch für jeden Programmierer, der die Hintergründe des Programmierens besser verstehen und seinen Stil verbessern will. Von der prosaischen Seite her gesehen ein nützliches Buch über Algorithmen, Datenstrukturen und gute Angewohnheiten beim Programmieren.

Kapitel 31

Linux Network Administrators Guide

Olaf Kirch

Ted Dawson

O'Reilly

Eine Einführung in das Netzwerkeln unter Linux. Dieses Buch behandelt alle wesentlichen Dinge, die ein Administrator braucht und gibt einen Einblick, wie ein Netzwerk überhaupt funktioniert. Es erklärt nicht alles bis zum letzen Bit (wer *alles* über TCP/IP wissen will, braucht andere Bücher), aber es beschreibt alles, was der Administrator im Alltag braucht: Die Grundlagen von TCP/IP, die Konfiguration der Hardware wie Netzwerkkarten oder Modems, Nameservices, Firewalls sowie das Network File System (NFS) und das gute alte UUCP.

Dann werden die Anwendungen des Netzwerks behandelt: Mail und News. Die Autoren erklären, wie E-Mail überhaupt funktioniert, und sie zeigen zwei wichtige MTAs im Detail: Sendmail und Exim. Postfix fehlt, aber darüber gibt es bei O'Reilly ein eigenes Buch. Von den Newsservern werden sowohl C News als auch INN besprochen. Zudem wird gezeigt, wie man einen Newsreader benutzt und konfiguriert.

Was fehlt, sind Webserver wie Apache. Aber auch darüber gibt es genügend eigene Bücher.

Kapitel 32

The Art of Computer Programming

Donald Knuth

Addison-Wesley

Hier sind sich wohl die meisten Programmierer einig, dass dieses Werk ein Klassiker ist. Es behandelt die Algorithmen und Datenstrukturen, die in vielen Programmen verwendet werden, und zwar in einer Tiefe, Ausführlichkeit und Gründlichkeit, die ihresgleichen suchen.

Das Buch besteht aus drei Bänden, die unter anderem folgende Themen abdecken: Suchen und Sortieren, Permutationen und Zufallszahlen. Außerdem werden Datenstrukturen wie Listen, Graphen und Bäume sowie die dazugehörigen Algorithmen behandelt.

Ein solches Buch sollte jeder Programmierer auf dem Schreibtisch haben (fanatische Programmierer können so etwas auch auf dem Nachttischchen haben). Eine Alternative zu Knuth und ebenfalls ein Klassiker sind die diversen Bücher von Robert Sedgewick, die ich allerdings nicht kenne (all meine Lesezeit zu diesem Thema habe ich mit den Bänden von Knuth aufgebraucht) und zu denen ich darum nichts sagen kann.

Zu diesem Buch habe ich auch eine emotionale Bindung, weil ich darin drei Fehler gefunden habe. Donald Knuth hat so viel Vertrauen in seine eigene Arbeit, dass er für jeden Fehler, den jemand in seinen Büchern findet, eine Belohnung aussetzt. Bei diesem Buch sind das pro Fehler 2,56 Dollar, was mir einen Scheck, handgeschrieben von Donald Knuth, eingebracht hat. Den habe ich natürlich nie eingelöst, sondern hinter Glas eingerahmt.

Kapitel 33

Don’t Make Me Think

Steve Krug

New Riders

Ein kleines, aber feines Buch über das Design von Webseiten. Hier geht es nicht um die Technik dahinter oder gar HTML, sondern darum, wie leicht und wie gut sich Anwender in einer Website zurechtfinden (Usability). Der Autor zeigt, wie die Elemente einer Webseite (Navigation, Banner, Text, Bilder) angeordnet sein müssen, damit Anwender sich dort sicher und mühelos bewegen können. Sie sollen schnell zu den Punkten gelangen, die dem Betreiber der Seite wichtig sind. Bei einer E-Commerce-Seite heißt das natürlich, dass sie möglichst schnell zum Einkaufen kommen. Wichtig ist auch, dass der Surfer oder die Surferin auf einen Blick erfassen kann, was der Sinn einer Web-Site ist: "Was haben die mir zu bieten?" "Was soll ich hier?"

Ein anderes Thema ist die Struktur der ganzen Site, also z. B., wie tief verschachtelt die einzelnen Teile sein sollen und welche Seiten zu Blöcken mit gleichem oder ähnlichem Design zusammengefasst werden sollen, so dass die Seite in Gruppen unterteilt wird, die durch ein einheitliches Layout zusammengehalten werden.

Sein Credo ist dabei "Don’t Make Me Think" ("Zwing mich bloß nicht zum Nachdenken"). Eine Website sollte so gestaltet sein, dass man mit minimalem (oder im besten Fall gar keinem) Denkaufwand erfassen kann, was wo ist, was wohin führt und was passiert, wenn man etwas anklickt.

Das Ganze veranschaulicht der Autor mit einer Fülle von Beispielen für gute und weniger gelungene Websites, darunter auch Klassiker wie Amazon, die darauf angewiesen sind, dass die Kundschaft auf ihnen leicht und bequem navigieren kann.

Wenn es wichtig ist, dass die Surfer auf Ihren Webseiten auf einen Blick sehen können, warum sie hier richtig sind (oder eben auch nicht), und dass sie leicht dahin gelangen, wo sie hinsollen (z. B. zum Einkaufen), dann sollten Sie dieses Buch lesen und seine Ratschläge beherzigen.

Kapitel 34

Applying UML and Patterns

Craig Larman

Prentice-Hall

Eine Einführung in objektorientierte Analyse und Design, unter intensivem Einsatz von UML. UML ist die Standardnotation für objektorientierte Softwareentwicklung. Die Stufen oder Zyklen eines Projekts, in dem objektorientierte Programmierung, Patterns, UML und Java zum Einsatz kommen, werden an einem ausführlichen Beispiel gezeigt. Dabei durchläuft das Projekt mehrere Iterationen, in denen das in den bisherigen Iterationen Gelernte genutzt wird, um das Ergebnis zu korrigieren und zu verfeinern. Das Beispiel exerziert alle Stufen in jeder Iteration des Projekts durch, von der Analyse über das Design bis zur Implementation. Die Implementation findet in Java statt. Dabei schwebt das Beispiel nicht die ganze Zeit in den luftigen Höhen von Design und Analyse, auch die kniffligen Details der Implementation werden behandelt.

Wie schon der Titel nahelegt, sind die Schwerpunkte dieses Buchs innerhalb der objektorientierten Softwareentwicklung zum einen UML und zum anderen Patterns. Das Buch zeigt, wie man UML einsetzt, und es behandelt alle wesentlichen Typen von UML-Diagrammen wie Klassendiagramme (welche die Struktur einer Klasse beschreiben, also ihre Eigenschaften und Methoden), Sequenzdiagramme (die zeigen, in welcher Reihenfolge welche Methoden einander aufrufen, also kurz gesagt, wie das Programm abläuft) und Kollaborationsdiagramme (die beschreiben, wie die Objekte zusammenarbeiten).

Während des Beispielprojekts kommen auch immer wieder Design Patterns zum Einsatz, und zwar einerseits Patterns, wie sie in Design Patterns beschrieben sind, andererseits sogenannte GRASP-Patterns. GRASP steht für General Responsibility Assignment Software Patterns (also ist der Ausdruck GRASP-Patterns streng genommen redundant, weil "Patterns" schon in GRASP enthalten ist). Diese Patterns beschreiben Rollen oder Veranwortlichkeiten von Objekten innerhalb eines objektorientierten Designs. Wenn das ein bisschen wolkig klingt, dann deshalb, weil es das auch ist. So beschreibt etwa das Pattern *Expert* "A class that has the information necessary to fulfill the responsibility" ("Eine Klasse, die die Informationen besitzt, die sie braucht, um ihre Aufgaben zu erfüllen"). Aha. Aber das trifft doch irgendwie auf jede Klasse zu? Aber hier heißt es: dranbleiben. Nach einer Zeit lichtet sich der Nebel und man versteht, wozu diese Patterns gut sind.

Darüber hinaus beschreibt das Buch auch Use Cases. Diese sind eine Prosabeschreibung eines (Teil-)Vorgangs bei dem Anwender des Systems involviert sind und sie werden idealerweise während der Designphase eingesetzt. Sie können mit Diagrammen beschrieben werden (in denen Strichmännchen mit Pfeilen verbunden werden) oder als Prosatext ("Der Kunde kommt an die Kasse. Der Kunde legt die Waren auf das Band. Die Kassiererin tippt den Preis für jede Ware ein..."). Use Cases sind auch ein elementarer Bestandteil des Extreme Programming (siehe auch Extreme Programming Explained).

Wem UML noch gar nichts sagt und wer eine angenehme Einführung in den Umgang mit UML ohne eine allzu steile Lernkurve sucht, der ist bei diesem Buch richtig.

Kapitel 35

The Cluetrain Manifesto

Rick Levine

Christopher Locke

Doc Searls

David Weinberger

Perseus Publishing

Nicht direkt ein Computerbuch, aber ein sehr lesenswertes Buch über das Leben in Zeiten des Internets. Wir sind nicht mehr nur dumme Hupferl, denen man "Kauf, Kauf!" (oder jetzt auch "Kaufen, Marsch, Marsch!") zuruft und die dann sofort parieren. Stattdessen versetzt uns das Internet in die Lage, direkt miteinander zu kommunizieren, uns wirklich zu informieren und unsere eigenen Entscheidungen zu treffen. Das glauben zumindest die Autoren.

Für die großen Konzerne ist das Internet hauptsächlich eine Fortsetzung des Fernsehens mit anderen Mitteln, bei der man einen passiven Anwender solange mit hohlem Marketing-Bla-Bla bombardiert, bis er von der Glotze aufsteht und einkauft. Aber so funktioniert das Internet nicht (nur), meinen die Autoren. Es hat den unerwarteten Nebeneffekt, dass die Leute direkt miteinander kommunizieren können, weil E-Mail allgegenwärtig ist, Newsgroups weit verbreitet sind und weil jeder mit wenig Aufwand eine Homepage einrichten kann, auf der er sagt, was er zu sagen hat. Und diese direkte Kommunikation unterläuft ganz zwangsläufig das Marketing-Geschrei der "Global Players".

Wenn eine Firma hinausposaunt, wie großartig, einmalig und unbedingt kaufenswert ihr Produkt XY ist, dann stehen dagegen eben auch zahlreiche Mails oder Newsgroup-Beiträge, in denen Leute schreiben "Ich hab’s gekauft. Der letze Sch...", oder gar Homepages, auf denen Leute ein Foto ins Web stellen, dazu den Text: "Das tolle XY, total kaputt, nachdem ich es zwei Wochen benutzt habe. Eine tolle Multimedia-Erfahrung, solange man keinen von den Knöpfen drückt. Leute, lasst die Finger davon!"

Und so macht das Internet das Leben zu einem Marktplatz, auf dem sich Aufrichtigkeit durchsetzt. Das meinen zumindest die Autoren. Wäre schön, wenn sie recht haben.

Kapitel 36

Expert C Programming

Peter van der Linden

Sun Soft Press/Prentice Hall

Dieses Buch beschreibt, wie C funktioniert, bis in die feinsten Details. Ein solches Detail ist beispielsweise, dass Arrays und Pointer nicht das gleiche sind, obwohl man sie in vielen Fällen austauschbar gebrauchen kann und die allermeisten C-Programmierer keinen Unterschied machen, weil sie sich dessen gar nicht bewusst sind. Van der Linden erklärt die Hintergründe, zeigt die Details, macht klar, wann man Arrays und Pointer gleichwertig behandeln kann und wann nicht. Zudem zeigt er, in welchen Fällen Arrays und Zeiger tatsächlich gleich sind, denn auch das gibt es verblüffenderweise. Bei der Lektüre des Buchs kann einem manchmal leicht schwindlig werden.

Programmieren kann ein haarige Angelegenheit sein, besonders in C, und deshalb illustriert der Autor seine Argumente mit ein paar kleineren Softwarepannen und größeren Softwarekatastropen aus der realen Welt.

Seine Exkurse über C lassen bei aller Kritik an den Ungereimtheiten dieser Sprache doch eine tiefe Sympathie für C erkennen. Ganz anders bei C++. Das hält er für den Tropfen, der das C-Fass zum Überlaufen bringt, für monströs und überkomplex, eine Meinung, der ich mich aus vollem Herzen anschließe. Seine prophetische Hoffnung ist, dass C++ nur dazu da ist, den Weg für etwas Besseres zu ebnen. Prophetisch ist diese Äußerung deshalb, weil er in einem seiner späteren Bücher, *Just Java*, zu dem Schluß kommt, dass Java genau das ist: Eine objektorientierte Sprache, die ihre Wurzeln (auch) in C hat und die die meisten schlechten Eigenschaften von C++ zu Gunsten von Einfachheit und Eleganz über Bord geworfen hat, eine Einschätzung, in der ich ihm ebenfalls weitgehend zustimme.

Wer sich ernsthaft für C interessiert (oder es sogar mag) profitiert enorm von diesem Buch. Außerdem: Es ist mit einer Menge Humor geschrieben, die Lektüre macht einfach einen Heidenspaß.

Kapitel 37

Mastering Algorithms with C

Kyle Loudon

O'Reilly

Dieses Buch ist eine Mischung aus zwei verschiedenen Arten von Büchern. Einerseits ist es ein Buch, das Algorithmen und Datenstrukturen beschreibt und zeigt, wie man sie implementiert (in diesem Fall in C), wie etwa in The Art of Computer Programming oder den Büchern von Sedgewick. Andererseits ist es ein Buch, das zeigt, wie man diese Implementation elegant und sauber mit Hilfe abstrakter Datentypen realisiert, ähnlich wie in C Interfaces and Implementations.

Es stellt die gängigen Datenstrukturen wie verkettete Listen, Stacks, Warteschlangen, Hashtables und Bäume vor; aber auch bekannte Algorithmen, wie etwa zur Verschlüsselung werden vorgeführt. Ein weiteres Thema sind Algorithmen für Graphen. Abgerundet wird das Ganze durch einige grundlegende Betrachtungen, etwa zu Zeigern oder zur Analyse von Algorithmen.

Was dieses Buch aus der Masse heraushebt, ist die Art und Weise, wie die Datenstrukturen implementiert werden. Nicht gehackt und gepfriemelt, sondern als Abstract Datatypes mit einem sauberen Interface. C-Programmierer können hier eine Menge lernen.

Kapitel 38

Writing Solid Code

Steve Maguire

Microsoft Press

Dieses Buch enthält wertvolle Tips, wie man stabilen Code schreibt, der gut zu warten ist. Der Schwerpunkt liegt auf dem Schreiben von Code, nicht auf der Programmentwicklung im Allgemeinen. Ein Thema, das sich durch das ganze Buch zieht, ist, Code so zu schreiben, dass er anderen Entwicklern, die mit diesem Code arbeiten müssen, möglichst wenig Rätsel aufgibt. Dieser Programmierer kann man auch selbst sein, wenn man ein Programm nach einigen Monaten wider Erwarten nochmals bearbeiten muss. Die Haltung "Nach mir die Sintflut!" ist allein schon deswegen gefährlich, weil sie auf einen selbst zurückschlagen kann. Zum Thema gut verständlicher Code zwei Beispiele aus dem Buch.

Der Autor plädiert dafür, das in C populäre Konstrukt

```
bedingung ? wenn wahr : wenn falsch
```

aufzugeben. Hand aufs Herz, welcher C-Programmierer muss nicht wenigstens kurz nachdenken, wie rum das Ganze funktioniert. Wie war das noch? Wenn der Ausdruck vor dem Fragezeichen wahr ist, dann wird die Anweisung vor dem Doppelpunkt ausgeführt. Oder war’s doch die Anweisung nach dem Doppelpunkt? Stattdessen soll man sich an if-else-Anweisungen halten, die einen zwar nicht als Insider oder Geek ausweisen, dafür aber auf Anhieb zu lesen sind:

```
if(bedingung) {
Anweisungen, wenn wahr;
}
else {
Anweisungen, wenn falsch;
}
```

Klar, jeder C-Programmierer, der sein Geld wert ist, versteht beide Ausdrücke. Auf lange Sicht erfordert aber der zweite Ansatz weniger Denkarbeit und senkt damit die Anzahl möglicher Fehlerquellen.

Eine weitere weit verbreitete Technik unter C-Programmierern ist es, die Vieldeutigkeit der Null auszunutzen. So steht die Null nicht nur für die Zahl, sondern sie hat in logischen Ausdrücken auch die Bedeutung "falsch". So weit, so gut. Darüber hinaus kann man die Null aber auch implizit als Ausdruck für den NULL-Zeiger verwenden, was Ausdrücke wie den Folgenden erlaubt:

```
int *zeiger = foo();
if(!foo) {
tu was;
}
```

Der Compiler macht hier aus der NULL eine Null, die dann für "falsch" steht. Maguire plädiert dafür, den Ausdruck explizit hinzuschreiben. Das ist nicht so knapp und cool, zeigt aber genauer was gemeint ist:

```
int *zeiger = foo();
if(foo != NULL) {
tu was;
}
```

Aus gutem Grund ist diese Konstruktion denn auch in Java verboten, dort ist der Shortcut lediglich für den Boolean-Wert erlaubt. Bei Referenzen (die ich hier etwas salopp mit Zeigern gleichstelle, was nur die halbe Wahrheit ist) muss man die NULL, die in Java klein geschrieben wird, explizit hinschreiben, also:

```
Foo foo = getSomeFoo();
if(foo != null) {
tu was;
}
```

Ich muss zugeben, dass ich mich auch nicht immer daran halte; die Formulierung ohne explizite NULL ist einfach ein gängiges Idiom in C. Es ist aber auf jeden Fall sinnvoll, mit der Nase auf solche Sachen gestoßen zu werden; ob man den Ratschlag beherzigt, kann man immer noch selbst entscheiden.

Wer es mit dem Programmieren ernst meint und sich und seinen Mitprogrammierern etwas Gutes tun will, sollte dieses Buch lesen. Wer nicht, der kann sich immer noch all die coolen Tricks abschauen und so in die Geek-Liga aufsteigen.

Kapitel 39

Code Complete

Steve McConnell

Microsoft

Ein Buch über die richtige Vorgehensweise beim Programmieren, das in eine dieselbe Kerbe schlägt wie Writing Solid Code oder The Practice of Programming. Dieses ist von den dreien das umfangreichste und es ist eine Fundgrube an Tips und Anregungen für guten Programmierstil. Die Themen reichen vom Formatieren des Quellcodes bis zum Vorgehen bei Projekten. Ausführlich werden Programmiertechniken beschrieben, so etwa das Programmieren mit Tabellen, mit dem man gewisse Typen von Programmen erheblich vereinfachen kann. Programmieren mit Tabellen ersetzt Programmcode durch Tabellen.

Ein Beispiel dafür ist die Berechnung einer Versicherungsprämie, die in einem Programm "zu Fuß" stattfinden könnte, indem man if-Anweisungen ineinanderschachtelt: "Wenn der Versicherte zwischen 20 und 30 Jahren alt ist, wenn er männlich ist und wenn er Raucher ist, dann beträgt der Versicherungsbeitrag 234 Euro im Monat". In diesem Fall ist die Berechnung hartcodiert, jede Änderung erfordert einen Eingriff in den Programmcode, im schlimmsten Fall an mehreren Stellen. Stattdessen kann man die Werte in eine mehrdimensionale Tabelle eintragen, auf die dann über Indices für Alter, Geschlecht, Raucher/Nichtraucher etc. zugegriffen wird.

Dies ist nur ein Beispiel für eine Fülle von Techniken und guten Angewohnheiten, die dieses Buch vorstellt und die es für jeden Programmierer unbedingt empfehlenswert machen.

Kapitel 40

Building Parsers with Java

Stephen John Metzker

Ein faszinierendes Buch darüber, wie man Parser baut. Normalerweise programmiert man Parser entweder von Hand (meistens ein elendes Gefummel, bis man ein Dateiformat und seine Syntax mit allen ihren Eventualitäten im Griff hat), oder man verwendet spezielle Programme wie Lex und Yacc (für C) beziehungsweise JavaCup (für Java), mit denen man eine Grammatik über eine eigene Sprache platzsparend beschreiben kann und die aus dieser Beschreibung einen passenden Parser erzeugen.

In diesem Buch wird die Sache einmal ganz anders angegangen und der Autor stellt einen objektorientierten Ansatz zum Parserbau vor. Jeder Teil des Parser-Baums wird durch ein Objekt repräsentiert. Das beginnt mit den Tokens, also den "Wörtern" der Sprache, die geparst werden soll. Auch die inneren Knoten, beispielsweise Wiederholungen oder Alternativen, werden durch Objekte repräsentiert. So verwendet man, wenn ein Wort ein oder mehrmals vorkommen soll, im Parser-Baum ein Objekt, das ein oder mehrere Vorkommen seiner Kind-Objekte akzeptiert. Darf genau eine Alternative aus einer Liste von Tokens vorkommen, so verwendet man ein Objekt, das diese Liste repräsentiert. Und so weiter.

Wie man schon aus dieser Beschreibung ersehen kann, ist das Buch recht gehaltvoll. Vorkenntnisse über Parser und objektorientierte Programmierung sollte man mitbringen. Dann aber ist das Buch ein faszinierende Lektüre, die das Innenleben eines Parsers richtig anschaulich macht. Ob man einen Parser im täglichen Programmiererleben tatsächlich auf diese Weise implementieren würde, ist eine ganz andere Sache, da werden die meisten auf bewährte Tools wie die oben erwähnten zurückgreifen.

Kapitel 41

Objektorientierte Softwareentwicklung

Bertrand Meyer

Hanser Verlag

Dieses Buch ist die deutsche Übersetzung des bei Prentice-Hall erschienenen Buchs *Object-oriented Software Construction*. Obwohl es vordergründig ein Buch über die Programmiersprache Eiffel ist (der Autor ist der Erfinder von Eiffel), werden hier nahezu alle Aspekte der objektorientierten Softwareentwicklung erschöpfend behandelt. Das beginnt mit Modularität und Komponenten und reicht über die richtige Konzeption von Schnittstellen zwischen Modulen bis hin zur Architektur von ganzen Softwaresystemen.

Großen Wert legt der Autor auf Kontrakte zwischen Modulen. In Eiffel sind sie folgerichtig sogar eingebaut, man kann den Kontrakt einer Klasse direkt in der Programmiersprache angeben. Was ist ein Kontrakt? Eine Art von Vertrag zwischen zwei Klassen, in dem sie sich gegenseitig zusichern, was sie liefern, wenn die Klienten der Klasse ihrerseits den Kontrakt erfüllen. Ein Beispiel: Eine Klasse könnte für ihre Methode `wurzel()` zusichern: Ich gebe Dir eine positive Zahl zurück, die hoffentlich die korrekt berechnete Wurzel des Eingabewerts ist. Dafür verlange ich von Dir, dass Du mir eine Zahl größer oder gleich Null als Eingabe gibst (weil ich keine Wurzeln aus negativen Zahlen ziehen kann; denn wir verwenden als Datentyp keine komplexen Zahlen). Man beachte, dass man in Kontrakten nur überprüfbare Tatsachen angeben kann. Dass ein Rückgabewert positiv ist, kann man vor der Rückgabe aus einer Methode prüfen. Dass der Wert die korrekt berechnete Wurzel eines Werts darstellt, könnte man nur prüfen, indem man die Wurzel berechnet. Dann hätte man aber eine weitere Methode, deren Korrektheit zu beweisen wäre, und da beißt sich die Katze in den Schwanz.

Trotzdem sind Kontrakte eine nützliche Angelegenheit; es gibt sie inzwischen für viele Sprachen als Erweiterung. Für Java gibt es beispielsweise Programme, die in Kommentaren definierte Kontrakte automatisch in Code umsetzen, der den Kontrakt prüft.

Die Sprache Eiffel hat sich nie in dem Maß durchgesetzt, wie sie es aufgrund ihrer konzeptionellen Reinheit (Diese Redewendung ist geklaut: In meinem Lieblingsfilm *Alien* antwortet der Roboter Ash auf die Frage, warum er das Alien in das Raumschiff gelassen und so die Besatzung dem sicheren Tod überlassen hat: "Ich bewundere die konzeptionelle Reinheit") verdient hätte, was wohl auch daran lag, dass sie nur mit teuren kommerziellen Entwicklungsumgebungen und Lizenzen zu erhalten war. Es gibt zwar eine freie GNU-Implementation von Eiffel, diese hinkt aber dem offiziellen Sprachstandard hinterher. So haben Sprachen das Rennen gemacht, die Eiffel eventuell konzeptionell unterlegen sind (das ist natürlich immer Ansichtssache), die aber frei und ohne Aufwand verfügbar sind.

Eiffel ist ein klassisches und tragisches Beispiel dafür, wie eine an sich gute Idee durch eine restriktive Lizenzpolitik kaputtgemacht werden kann. Das ist auch eine deutliche Warnung für die aktuelle Diskussion um Softwarepatente: Wenn man all seine tollen Ideen hinter goldenen Mauern aus teuren Lizenzen einsperrt, kann es sein, dass sie niemand mehr haben will.

Trotz alledem ist das Buch absolut lesenswert, weil es über Eiffel hinaus ein großes Kompendium der objektorientierten Softwareentwicklung ist, das zudem so geschrieben ist, dass man die Ideen und Konzepte leicht auf andere Programmiersprachen übertragen kann.

Kapitel 42

Cascading Style Sheets 2.0 Programmers Reference

Eric Meyer

Osborne/McGraw-Hill

Die passende Ergänzung zu Eric Meyer on CSS vom gleichen Autor. Diese Referenz beschreibt alle wesentlichen Elemente von CSS ausführlich. Neben den bekannten Elementen für das Layout von Webseiten werden auch Elemente für Stylesheets, welche die Druckausgabe eines Dokuments steuern, sowie Stylesheets für die akustische Wiedergabe von Webseiten behandelt. (Damit ist nicht Musik auf Webseiten gemeint, sondern die akustische Ausgabe von Webseiten, beispielsweise durch Screen-Reader für Sehbehinderte.)

Darüber hinaus beschreibt das Buch die Prioritätsregeln, die bestimmen, welche Regel für ein bestimmtes Element des XML- oder HTML-Dokuments angewandt wird. Diese sind durchaus ein bisschen kompliziert und bedürfen der Erläuterung. Schließlich enthält das Buch Angaben, welcher Browser ein bestimmtes CSS-Feature auch tatsächlich implementiert. Denn grau ist alle Theorie: So schön und nützlich viele Elemente von CSS sind, manche Browser implementieren sie falsch oder einfach gar nicht.

Fazit: Eine sehr nützliche und nicht zu teure Referenz für alle, die mit CSS arbeiten.

Kapitel 43

Eric Meyer on CSS

Eric Meyer

New Riders

Faszinierend, was man mit CSS alles machen kann: Vom mehrspaltigen Layout über aufgepeppte Links bis hin zur Druckausgabe einer Webseite. Einen Vorgeschmack davon gibt es auf der Webseite zum Buch. Dieses Buch zeigt anhand von Beispielprojekten, wie man CSS einsetzt, um Webseiten zu gestalten. Das beginnt beim einfachen Ausrichten von Elementen auf einer Webseite (eine Aufgabe, die vor CSS mit Hilfe von geschachtelten Tabellen gelöst wurde) und führt bis zu Dingen wie animierten Links (Links, die ihr Aussehen verändern, wenn der Mauszeiger über ihnen schwebt). Außerdem sieht man, wie Webseiten mit CSS für die Druckausgabe aufbereitet werden; dies ist oft eine schnelle und günstige Alternative zur doch recht schwerfälligen und aufwendigen Konvertierung per XSLT. Man muss nur das Druck-Stylesheet auswählen und das Dokument aus dem Browser heraus drucken.

Kapitel 44

Enterprise Java Beans

Richard Monson-Haefel

O'Reilly

Hier geht es um Enterprise Java Beans. Sie sind nicht so verbreitet, wie der Hype, der bei ihrer Entstehung herrschte, nahegelegt hätte. Dennoch haben sie ihren festen Platz in der Softwareentwicklung mit Java.

Kurz gesagt ist eine Enterprise Java Bean eine Komponente, die in einem Enterprise-Server (Er muss natürlich Enterprise-Server heißen, ohne das Buzzword *Enterprise* wäre er für das Marketing wertlos) lebt und von diesem über das Netzwerk an Client-Software ausgeliefert wird. Um die Herstellung der Enterprise Bean aus einer Datenbank (im Tech-Speak: Serialisierung und De-Serialisierung) und die sonstige Infrastruktur kümmert sich der Server.

Dieses Buch ist das umfassende Werk zum Thema Enterprise Beans. Es zeigt die Implementierung und die Anwendung. Es geht zudem auf die Hintergründe ein, etwa darauf, wer bei einer Bean den aktuellen Zustand speichern soll, die Bean (was dann treffend mit *Bean Managed Persistence* bezeichnet wird) oder der Server (*Container Managed Persistence*), und untermauert das Ganze mit etlichen Beispielen. Ebenfalls ausführlich erläutert wird der Unterschied zwischen Entity Beans, die Daten speichern und im Wesentlichen die Repräsentation einer Datenbanktabelle als Java-Objekt sind, und Session Beans, die einen Vorgang in der Software (neudeutsch: Workflow) beschreiben.

Kapitel 45

HTML & XHTML

Chuck Musciano

Bill Kennedy

O'Reilly

Der Untertitel sagt es schon: Der perfekte Begleiter durch den Dschungel der Elemente in HTML und XHTML. Jedes Element wird besprochen, mitsamt seinen Attributen und mit Beispielen, wie man es einsetzt. Darüber hinaus gibt das Buch Anleitungen, wann man welches Element verwenden kann und soll. Zudem wird für jedes Element angegeben, welcher der Browser Netscape (bzw. Mozilla) und Internet Explorer es unterstützt. Das ist natürlich besonders wichtig bei solchen Elementen, die nur ein Browserhersteller implementiert hat, wie etwa das `<marquee>`-Element, mit dem nur der Internet Explorer etwas anfangen kann.

Die Elemente werden im Buch nicht einfach in alphabetischer Reihenfolge vorgeführt, sondern anhand der Aufgaben, die sich beim Schreiben von HTML stellen, eins nach dem anderen vorgestellt. Das beginnt bei einfachem Text, für den Elemente wie `<b>`, `<i>` oder `<p>` zum Einsatz kommen. So geht es dann mit steigender Komplexität weiter, über Listen und Formulare zu Tabellen. Das hat den Vorteil, dass man für jedes Element gleich sieht, wozu es gut ist und wann man es anwenden kann. Ausführlich beschrieben werden auch Unterschiede und Gemeinsamkeiten zwischen HTML und XHTML.

Fazit: Wenn man an Webseiten arbeitet, sollte dieses Buch in Griffweite sein; es bietet alle wichtigen Informationen über HTML und XHTML ausführlich und übersichtlich.

Kapitel 46

Games, Diversions & Perl Culture

Jon Orwant

O’Reilly

Spaß mit Perl. Neben einigen wissenschaftlichen Programmen vor allem Skurrilitäten, die typisch für Perl sind: One-Liners, also kurze Programme, die (idealerweise) in eine Zeile passen. Obfuscated Perl, also Perl-Programme, die so undurchschaubar wie möglich sind.

Kapitel 47

Unix Power Tools

Jerry Peek

Tim O'Reilly

Mike Loukides

O'Reilly

Eine Wunderkiste voller Unix-Tricks. Hier geht es nicht um Programmieren in Sprachen wie C oder C++, sondern um das Arbeiten mit der Shell und bekannten und beliebten Unix-Tools wie Sed, Awk, Grep, Find oder Perl. Die Autoren greifen tief in die Trickkiste und zeigen auf ca. 1000 Seiten Tricks zu den Themen Benutzerverwaltung, Systemadministration, Netzwerkadministration etc. Sie gehen ausführlich auf die Shell ein und behandeln sowohl Grundlagen als auch sehr fortgeschrittene Tricks. Ausführlich widmen sie sich den Regular Expressions, die für das Skripten unentbehrlich sind. Weitere Themen sind: Umgebungsvariablen, wie man die History der Shell ausnutzt, wie man Jobs kontrolliert (in den Vordergrund holt, in den Hintergrund schickt, anhält, stoppt und wieder startet), Redirection (mit der die Shell die Ausgabe eines Programms auf die Eingabe eines anderen schickt, was einen großen Teil der Magie der Shellprogrammierung ausmacht) und vieles mehr.

Die Autoren machen klar, was die Philosophie von Unix ist: Viele kleine Tools, durch Shellskripte verbunden und immer neu kombiniert. Die Tools kommunizieren durch Text miteinander, so dass der Mensch immer mitlesen darf und eine Chance hat zu verstehen, was da gerade abläuft.

Dieses Buch ist sehr empfehlenswert für jeden, der mit Unix arbeitet. Man kann aber auch einfach nur darin schmöckern und sich an den für Unix so typischen Shell-Hacks erfreuen.

Kapitel 48

Perl 6 Essentials

Allison Randal

Dan Sugalski

Leopold Tötsch

O'Reilly

Meine gespaltene Meinung zu Perl habe ich ja schon kundgetan (in der Besprechung zu Programming Perl). Aber egal, wie man zu Perl 5 steht, mit Perl 6 tut sich eine Menge und wenn es dann wirklich verfügbar sein wird, gehört Perl 6 je nach Standpunkt entweder zu den aufregendsten oder zu den nervenaufreibendsten Dingen in der Computer-Welt.

Dieses kleine Büchlein gibt einen Überblick über das neue Perl und über die Virtual Machine names Parrot, auf der Perl dann laufen soll. Dabei wird Parrot recht viel Raum eingeräumt, viele Leser dürfte Parrot nicht so sehr interessieren, sie würden wohl lieber noch mehr über die Syntax von Perl 6 erfahren.

Um mit der aktuellen Entwicklung von Perl 6 Schritt zu halten, erscheint einmal pro Jahr eine Neuauflage. Fazit: Wer, aus welchen Gründen auch immer, über Perl 6 auf dem Laufenden bleiben will, findet hier die wichtigsten Informationen auf knappem Raum, aber in fundierter Form.

Kapitel 49

The New Hackers Dictionary

Eric S. Raymond

The MIT Press

Ein Wörterbuch des Hacker-Vokabulars. Hacker meint hier nicht einen Hacker in dem Sinn, in dem das Wort heute meist gebraucht wird, nämlich jemand ganz bösen, der in Computersysteme eindringt und da alles kaputtmacht. Hacker im Sinne dieses Buches ist jemand, der kreativ mit dem Computer umgeht und versucht, ihn auch auf Arten und Weisen zu nutzen, die ursprünglich nicht vorgesehen waren.

Die Einträge in diesem Buch sind über Jahrzehnte gesammelt worden und Eric Raymond hat sie schließlich in Buchform herausgebracht.

Die Hackerszene hat einen ganz eigenen Jargon, meist ironisch, manchmal bissig, mit dem sie die Begriffe aus ihrer Umgebung belegt. Meist geht es dabei um Hard- und Software, aber auch die Menschen, mit denen Hacker zu tun haben, wie Bürokraten oder Manager, bekommen ihr Fett weg.

Alles, was in diesem Buch steht, kann man auch im Internet finden. Trotzdem lohnt sich das Buch, um an einem verregneten Herbsttag oder im Urlaub darin zu schmöckern und stillvergnügt vor sich hinzukichern (Lesetip: Der Eintrag zum Stichwort "blinkenlights").

Kapitel 50

Body Types

Günter Schuler

Smartbooks

Wer viel mit einer Textverarbeitung arbeitet, ein Satzsystem wie TeX oder LaTeX benutzt oder XML nach PDF oder PostScript konvertiert, wird sich irgendwann für Fonts interessieren. Er oder sie wird sich fragen, welche Schrift für welchen Zweck die richtige ist und warum sie die richtige ist.

Dieses Buch beantwortet solche Fragen. Es beschreibt die Geschichte der Schriften vom Römischen Reich bis heute, wobei Klassiker wie Garamond, Helvetica oder Times ausführlich gewürdigt werden. Auch die Rückwendung zur Fraktur im Dritten Reich wird gezeigt (und gebührend durch den Kakao gezogen). Anschließend werden die Merkmale einer Schrift im Detail vorgestellt: Was sind Serifen, was ist eine kalligraphische versus eine konstruierte Schrift? Woran erkennt man eine amerikanische Schrift? (An den im Vergleich zu den Großbuchstaben relativ hohen Kleinbuchstaben, der Unterschied zwischen Groß- und Kleinbuchstaben ist nicht so ausgeprägt wie bei einer europäischen Schrift.) Die Schriften werden klassifiziert, in Renaissance-Schriften, klassizistische Schriften, und schließlich die modernen, oft serifenlosen Schriften, die der Zunft am Anfang so suspekt waren, dass sie als "grotesk" bezeichnet wurden, was etliche von diesen Schriften noch heute im Namen tragen. Typisch für die heutige Zeit sind Schriftclans und Multiple-Master-Schriften, bei denen eine Schriftfamilie verschiedene Schrifttypen wie Serif, Serifenlos, und evtl. Zierschriften unter einem Dach vereint.

Den Abschluß des Buchs bildet ein Katalog der Schriften, nach den großen Einheiten wie klassizistisch, Renaissance und Schrift-Clans gegliedert, der mit ausführlichen Schriftproben garniert ist. Diese zeigen die Schriften sowohl in Musterbuchstaben als auch mit Blindtextproben, die demonstrieren, wie eine Schrift im Gebrauch aussieht. Sofern der Font diese enthält, werden auch die kursiven und fetten Varianten der Schrift gezeigt.

Fazit: Ich hätte mir das Buch ein bisschen allgemeinverständlicher gewünscht; man merkt, dass es vor allem auf den gewieften Werbefachmann zielt, dem Begriffe wie Durchschuss nur ein müdes "Klar weiß ich, was das ist" entlocken. Es enthält zwar ein Glossar, aber als interessierter Laie würde man einfach gern ein wenig mehr "an die Hand genommen" werden. Trotzdem ist es für jeden, den es interessiert, in welcher Schrift seine Texte gedruckt werden, und der eine Leitlinie bei der Entscheidung für oder gegen bestimmte Schriften braucht, ein sehr wertvolles Buch.

Anmerkung: 2004 war das Buch ziemlich günstig bei Zweitausendeins zu erwerben, deutlich reduziert gegenüber dem offiziellen Preis. Das kann sich natürlich inzwischen geändert haben.

Kapitel 51

Programming Ruby

David Thomas

Andrew Hunt

Ruby ist im Trio der Skriptsprachen Perl, Python, Ruby die jüngste und es enthält (wie viele sagen, die jeweils besten) Elemente seiner beiden Vorgänger. Ruby hat alle Zutaten, die eine nützliche Skriptsprache braucht, und dabei eine einfache und klare Syntax. Ruby ist *orthogonal*, was für den alltäglichen Gebrauch soviel bedeutet wie: Wenn man nicht weiß, wie etwas funktioniert, dann funktioniert es in etwa so, wie man sich das selbst auch gedacht hätte (während manches in Perl ganz anders funktioniert, als man es sich selbst gedacht hätte.)

Von Perl hat es die eher konventionelle Syntax. Während bei Python die Tiefe der Einrückung die Logik des Programms bestimmt, sind bei Ruby Blöcke und Programmlogik durch Schlüsselwörter wie `do` und `end` markiert. In Perl erfüllen geschweifte Klammern denselben Zweck. Von Perl hat Ruby auch die "speziellen Variablen", wie etwa `$~` für den Match der zuletzt ausgeführten Regular Expression, geerbt. Zu jeder dieser Variablen gibt es aber ein Objekt (in diesem Fall `MatchData`), über das man auf dieselben Daten zugreifen kann.

Von Python hat Ruby die saubere Syntax geerbt, ohne dass in Ruby die Einrückungstiefe der Zeilen die Syntax des Programms bestimmt. So schön Python sonst ist, dieses Feature hat schon etliche Programmierer (mich eingeschlossen) an den Rand des Wahnsinns getrieben.

Warum programmieren dann eigentlich nicht alle in Ruby? Dafür gibt es mehrere Gründe: Erstens gibt es eine riesige Menge an Skripten und Programmen in Perl und Python, die klag- und problemlos ihren Dienst tun, und wie heißt es so schön: "Never change a winning team". Zweitens gibt es für Python und insbesondere für Perl eine riesige Menge an Bibliotheken für jeden denkbaren Zweck, vom Zugriff auf Netzwerk-Sockets bis zum Auslesen der Exif-Daten aus digitalen Fotos. Da kann Ruby noch nicht mithalten, einfach weil die beiden anderen Sprachen einen Vorsprung von mehreren Jahren haben. Ein dritter Grund ist natürlich, dass bestimmt nicht jeder Ruby mag. Programmiersprachen sind auch Geschmackssache, Ruby macht da keine Ausnahme, und nicht jedem wird Ruby sympathisch sein.

Das Buch der Pragmatic Programmers David Thomas und Andrew Hunt ist eine sehr gute Einführung in Ruby und dazu eine gute Referenz, in der man die zahlreichen Klassen in Ruby beschrieben findet. Wer Gefallen an Ruby gefunden hat und einen noch tieferen Einblick in die Sprache gewinnen will, sollte zusätzlich noch The Ruby Way lesen.

Kapitel 52

Delphi Component Design

Danny Thorpe

Addison-Wesley

Wenn schon Delphi, dann wenigstens mit diesem Buch als Unterstützung. Wenn Sie daraus schließen, dass ich Delphi nicht mag, liegen Sie richtig. Es gibt allerdings immer noch eine Menge Programmierer, die mit Delphi arbeiten und für die ist dieses Buch der richtige Begleiter, wenn sie mehr als ganz einfache Prograämmchen schreiben wollen.

Das Buch zeigt Komponenten sowohl von der Seite des Anwenders, der eine Komponente lediglich von der Palette zieht und in sein Programm einbaut, als auch von der Seite des Komponentenentwicklers, der selbst Komponenten baut. Dazu muss er Bescheid wissen über Property Editors, Component Editors und OLE und COM; all das bietet das Buch in aller Ausführlichkeit und mit Beispielen unterfüttert, und dazu in einem leichten, lockeren und augenzwinkernden Ton.

Kapitel 53

The Elements of Java Style

Allan Vermeulen

Scott W. Ambler

Greg Bumgardner

Eldon Metz

Trevor Misfeldt

Jim Shur

Patrick Thompson

Cambridge University Press

Was Strunk und White mit *The Elements of Style* für die englische Sprache und Kernighan und Plauger mit *The Elements of Programming Style* für das Programmieren im Allgemeinen getan haben, tut dieses Büchlein für Java. Es ist eine Sammlung von Richtlinien, Konventionen und guten Angewohnheiten, die dabei helfen, guten, lesbaren und verständlichen Java-Code zu schreiben. Das beginnt bei so grundlegenden Dingen wie der Benennung von Variablen und Methoden und der Art und Weise, wie Code formatiert wird. Weiter geht es darum, wie man Code richtig kommentiert. Über die Konventionen, an die man sich dabei halten sollte, kann man trefflich streiten, wichtig ist vor allem, dass sich in einem Programmierprojekt alle an dieselben Konventionen für das Benennen und Formatieren halten.

Schließlich geht es zu den Programmiertechniken, etwa dem Tip, Klassen und Methoden möglichst klein zu halten, oder dem Hinweis, Anweisungen, die bewusst leer sind, durch einen Kommentar als solche zu kennzeichnen, wie im folgenden Code-Schnipsel:

```
    // Ignoriere fuehrende Leerzeichen.
    while((c = reader.read()) == SPACE) {
    // Ignorieren.
    }
```

Gerade diese Tips enthalten viele Dinge, die einleuchtend sind, auf die man aber selbst nicht so leicht kommt.

Dieses Buch ersetzt nicht die Lektüre eines "großen" Buchs über guten Programmierstil wie etwa Writing Solid Code oder Code Complete. Als Einstieg ist es jedoch gut geeignet und dank seines geringen Umfangs und seines niedrigen Preises kann man es bei nicht allzu großen Projekten an alle Programmierer verteilen, so dass alle die selben Programmierkonventionen ständig vor der Nase haben und sich daran halten können.

Kapitel 54

Programming Perl

Larry Wall

Tom Christiansen

Jon Orwant

O'Reilly

"There's More Than One Way To Do It" ist das Motto dieses Buchs und das Motto von Perl. Was für die einen das Höchste ist, die Freiheit, Dinge auf verschiedene Weise zu tun, ist für die anderen die Hölle: In einer vor Features überquellenden Programmiersprache wenigstens einen Weg zu finden, es richtig zu machen. Da kann es dann vorkommen, dass man das Perl-Motto fluchend zu "Is There At Least One Way To Do It Right?" ummodelt.

Ich selbst kann mich nicht entscheiden. Ich bin abwechselnd fasziniert von dem barocken Reichtum an Syntax und Semantik in Perl, dem Ansatz, eine Programmiersprache wie eine gesprochene Sprache anzulegen, mit bewussten Mehrdeutigkeiten, von der "Magie" hinter den Kulissen (laut dem Tcl-Wiki enthalten die Man-Pages zu Perl 493 Vorkommen des Wortes "magic"), und dann wieder abgeschreckt von soviel Mehrdeutigkeit. Dann flüchte ich mich zu einfacheren Sprachen wie etwa Ruby.

Unabhängig davon, wie man selbst zu Perl steht, ist das Buch absolut lesenswert, erfüllt von einem augenzwinkernden Humor, der das Lesen zum Vergnügen macht. Und egal, was man von Perl hält oder ob man überhaupt schon einmal eine Zeile Perl programmiert hat: Ein ernsthafter Programmierer sollte einfach wissen, worum es bei Perl geht. Man muss nicht in Perl programmieren können oder wollen, aber man sollte ein wenig Perl verstehen, da man immer wieder auf Perl-Skripte trifft. Perl ist nicht jedermanns Sprache, aber es ist zu verbreitet, um es zu ignorieren.

Kapitel 55

DocBook

Norman Walsh

Leonard Muellner

O'Reilly

DocBook ist eine der wichtigsten XML-DTDs (bzw. Schemas). Es ist insbesondere für technische Dokumente wie Handbücher oder Computerbücher gedacht. Ein großer Vorteil von DocBook ist, dass es frei von Lizenzgebühren ist. Auch das Buch, das Sie gerade in lesen, ist in DocBook geschrieben.

Das Buch von Norman Walsh ist eine umfangreiche Referenz zu DocBook. Jedes Element wird ausführlich behandelt, mitsamt Beispielen, wie man die Elemente verwendet. Zudem wird für jedes Element angegeben, wo man es verwenden kann, denn nicht jedes Element kann überall in einem DocBook-Element auftauchen. So gehört beispielsweise ein `<title>` immer zu einem anderen Element wie einem Kapitel oder einer Tabelle. Der Anhang zeigt die lange Liste der Entities in DocBook, mit denen man auch ungewöhnliche Sonderzeichen darstellen kann. Wenn man Dokumente in DocBook schreibt, ist es gut, dieses Buch immer in Griffweite zu haben.

Kapitel 56

Vim Ge-Packt

Reinhard Wobst

mitp-Verlag

Ein mässig dickes Buch in handlichem Format. Es ist eine Referenz für Vim, die erweiterte Variante des klassischen Kommandozeileneditors Vi. (Dabei ist Vim abwärtskompatibel zu Vi, alles, was Vi kann, kann Vim auch.) Das Buch betet nicht einfach alle Befehle des Editors herunter, sondern versucht, dem Anwender dabei zu helfen, dass er oder sie sich möglichst schnell in Vim zurecht findet und sich dann selbst helfen kann. Deshalb steht eine ausführliche Einführung in das Hilfe-System von Vim ganz am Anfang.

Da das Buch nicht lediglich die Vim-Befehle seitenlang auflistet (natürlich gibt es auch hier Listen mit Befehlen, aber nur, wenn es wirklich von Vorteil ist, sie auf Papier nachschlagen zu können statt sie im Hilfe-System suchen zu müssen), bleibt Platz für etliche weiterführende Themen wie die Skriptsprache des Vim, Faltungen (das Zusammenfassen von Code-Blöcken wie etwa Funktionen zu einer einzigen Zeile, um Platz auf dem Bildschirm zu sparen) oder die automatische Syntaxerkennung, mit der Vim anhand der Endung einer Datei oder anhand anderer Merkmale erkennt, um welchen Typ von Datei (Java-Code, Shell-Skript, HTML etc.) es sich handelt.

Kapitel 57

Decline and Fall of the American Programmer

Edward Yourdon

Prentice Hall

Der Titel führt etwas in die Irre. Der größte Teil des Buchs handelt einfach von Softwareentwicklung und Projektmanagement und bietet einen guten Überblick über Techniken und Methoden der Softwareentwicklung. Der Aufhänger ist aber trotzdem die Globalisierung, die dazu führt (na, wer hat den Spruch noch nie gehört?), dass woanders – vorzugsweise in Indien – billiger, besser und produktiver gearbeitet wird, weshalb sich der amerikanische Programmierer ganz besonders anstrengen muss (und im übrigen laut Yourdon natürlich völlig überbezahlt ist).

Mit dieser These als Ausgangspunkt stellt der Autor Methoden der Softwareentwicklung vor, die dem amerikanischen Programmierer zumindest ein wenig helfen sollen, Anschluss an andere Produktionsstandorte zu finden. Die Methoden sind: Objektorientierte Programmierung, CASE (Computer-Aided Software Enginering), Software Metrics etc. Das Thema "Silver Bullets" wird auch in diesem Buch behandelt und der Autor kommt ähnlich wie Frederic Brooks in The Mythical Man Month zu dem Schluss, dass es keine wirkliche Silver Bullet gibt, aber doch deutliche Verbesserungen der Produktivität durch verschiedene Techniken. Yourdon geht auch auf Peopleware ein, also den menschlichen Aspekt der Softwareentwicklung, und beschäftigt sich auch mit dem Thema, wie man Mitarbeiter richtig einstellt und feuert. Wenn das jetzt zynisch klingt, dann deshalb, weil es das ist. Yourdon handelt das Thema ab wie jede der vielen anderen technischen Schwierigkeiten der Softwareentwicklung, eine Sache, die man eben erledigen muss. Diesem Buch fehlt ebenso wie Rise and Resurrection of the American Programmer völlig der warme, humane Ton, der andere Bücher über Softwareentwicklung wie The Mythical Man Month oder Pragmatic Programmer durchzieht.

Trotz all dem ist das Buch ein kompakter Überblick über die Softwareentwicklung und es bietet dem "gewöhnlichen Programmierer" auch einen oft deprimierenden Einblick in die Gedankenwelt der "Entscheider". Ob einem diese Welt gefällt oder nicht, man sollte wissen, wie die Entscheider denken, dann ist man besser informiert und nicht so überrascht, wenn man selbst mit den Konsequenzen dieser Entscheidungen konfrontiert wird.

Kapitel 58

Rise and Resurrection of the American Programmer

Edward Yourdon

Prentice Hall

Auch hier führt der Titel etwas in die Irre. Der Themenbereich ist ähnlich wie bei Decline and Fall, aber dieses Buch strahlt etwas mehr Hoffnung aus. Es stellt Technologien vor, die der (amerikanischen) Softwareentwicklung wieder auf die Beine helfen könnten. Dazu gehören Java und das Internet im Allgemeinen. Ein wesentlicher Aspekt ist das Konzept der "Good Enough Software": Software muss nicht perfekt sein, sondern eben nur "gut genug", das spart Geld, Zeit und Entwicklungskosten. Wenn man sich die Qualität vieler Programme anschaut, kann man tatsächlich zu dem Schluss kommen, dass viele Softwarehersteller diesem Ruf gefolgt sind.

Das Kapitel über Java erweckt gemische Gefühle. Auf der einen Seite ist es schön, dass Yourdon das Potential dieser Programmiersprache erkannt hat. Auf der anderen Seite enthält das Kapitel etliche Patzer; man merkt, dass Yourdon Consultant und kein Programmierer ist und dass er auf der technischen Seite nicht wirklich trittsicher ist. So behauptet er, eine Klasse in Java sei ein Applet ("a ’class’ is an entire Java applet") (umgekehrt wird ein Schuh daraus: Ein Applet ist eine Klasse) und er behauptet, die Entwicklung von Java habe sich in zwei Äste gespalten, einen mit normalem Java und einen mit Java light, und dieses Java light sei JavaScript. Dabei beginnt so gut wie jedes JavaScript-Buch mit der Warnung: Lassen Sie sich nicht vom ähnlichen Namen irritieren, Java und JavaScript sind zwei grundverschiedene Dinge.

Trotz allem ist auch dies ein guter Überblick über Methoden, Technologien und die Peopleware der Softwareentwicklung und wie Decline and Fall ein ernüchternder Einblick in die Gedankenwelt vieler, die in der Softwareentwicklung maßgebliche Entscheidungen treffen.

Kapitel 59

Death March

Edward Yourdon

Prentice Hall

Traurig, aber wahr. Alle klassischen Fehler, die man beim Projektmanagement machen kann, in einem kleinen Buch versammelt. Auch wenn es einen ziemlich frustrieren kann: Dieses Buch sollte jeder Programmierer gelesen haben, damit er weiß, warum das mit ihm passiert, was da gerade mit ihm passiert, wenn er jemals in so ein Projekt hineingerät (und so gering soll die Wahrscheinlichkeit ja nicht sein).

Unter einem Death March versteht Yourdon ein Projekt, in dem ein oder mehrere für die Machbarkeit des Projekts entscheidenden Faktoren um 50 Prozent oder mehr unterschritten bzw. überschritten werden, immer im Vergleich zu einer rationalen Schätzung. Yourdon geht natürlich darauf ein, was eine rationale Schätzung ist und wie man sie erstellt. Die 50-Prozent-Regel kann dann bedeuten: Es steht weniger als die halbe Zeit zur Verfügung, die man eigentlich bräuchte. Es sind nur halb so viele Programmierer am Projekt beteiligt wie nötig. Das Budget ist nur halb so groß, wie es sein müßte. Oder, besonders beliebt, das Projekt wächst wegen unzureichender Planung, Versprechen des Marketing (das diese selbst nicht erfüllen muss und sie deshalb leicht geben kann) oder immer weiterer Sonderwünsche des Kunden auf den doppelten Umfang, wobei natürlich die Ressourcen wie Leute (Peopleware), Budget oder Zeit nicht verdoppelt werden. Förderlich für den Death March ist laut Yourdon auch die unter Managern und auch Programmierern weit verbreitete Ansicht "Real programmers don’t need sleep" ("Echte Programmierer brauchen keinen Schlaf").

All dies führt zu einem Death March, dessen Konsequenzen wie maßlose Überstunden, ein Management, bei dem es keiner gewesen sein will und stattdessen jeder mit dem Finger auf den anderen zeigt (Programmierer wie Manager) sowie allgemeine Panik Yourdon ausführlich, ja fast schon genüsslich schildert.

Er geht auch darauf ein, wie man einen Death March erkennt, was man dagegen tun kann und wann eine Kultur des Deatch March sinnvoll sein kann, z. B. in einem Startup, in dem nach einer überschaubaren Phase der Überarbeitung und Panik eine Belohnung erfolgt (natürlich für alle Beteiligten, nicht nur die Manager), etwa in Form eines Börsengangs. Eine solche Belohnung kann laut Yourdon die Strapazen im Nachhinein rechtfertigen.

Kapitel 60

Designing with Web Standards

Jeffrey Zeldman

New Riders

Jeffrey Zeldman plädiert dafür, Webseiten mit Webstandards zu erstellen: XHTML, CSS, JavaScript. Recht hat er. Das Layout von Seiten mit CSS zu regeln, ist deutlich einfacher und übersichtlicher, als Tabellen mehrfach ineinander zu verschachteln und Spacer-Gifs zu verwenden. (Spacer-Gifs sind unsichtbare Bilddateien vom Typ GIF, die eine Größe von 1 mal 1 Pixeln haben und die in Webseiten eingebaut werden, um das Layout im Detail zu arrangieren.)

Seiten, die mit diesen Techniken geschrieben werden, sehen mindestens so gut aus wie Seiten, die mit Tabellen und Spacer-Gifs hingetrickst werden. Ihr großer Vorteil ist, dass sie auch dann funktionieren, wenn der Browser (z. B. ein älteres Modell, der Text-Browser Lynx oder der Browser eines PDA) das ausgefeilte CSS-Layout nicht darstellen kann. Dann wird zwar nur eine nackte HTML-Version angezeigt, die aber wenigstens benutzbar bleibt, während mit komplizierten Tabellen-Layouts konstruierte Seiten auf solchen Geräten oft völlig zusammenbrechen. Generell gilt: Besser eine benutzbare Seite ohne tolles Layout als eine Seite mit irrsinnig schickem Layout, die leider nicht benutzbar ist (weil beispielsweise Links nicht mehr erscheinen oder sich Elemente gegenseitig überdecken). Ein weiterer Vorteil dieses Vorgehens: Man spart sich den Aufwand, die Seiten an jedes Browser-Modell einzeln anzupassen.

Nachteil: Obwohl Web-Standards Standards sein sollten, klappt das in der Praxis nicht immer, ein Problem, das bereits von HTML sattsam bekannt ist. Etliche Features, die eigentlich jeder kompatible Browser gleich implementieren sollte, werden verschieden interpretiert. Deswegen ist man gut beraten, nur einen Teil von CSS etc. einzusetzen. Auch auf solche Probleme geht Jeffrey Zeldman in diesem Buch ein.

Kapitel 61

Kolophon

Diese Buch wurde in XML geschrieben, die DTD ist DocBook. Für das Layout und zum Ausdruck wurde es mit dem XMLmind FO Converter von XMLmind in RTF umgewandelt. Dieses RTF wurde dann in OpenOffice importiert und formatiert. Dieses RTF wurde dann in OpenOffice importiert, anschließend als einfacher Text abgespeichert, erneut in OpenOffice geladen und dort von Hand formatiert. Dieser Umweg war nötig, weil eine in OpenOffice importierte RTF-Datei sich nicht ohne Probleme weiterverarbeiten lässt.

Die Orchidee auf dem Umschlag ist eine Phalaenopsis oder eine Doritaenopsis. Genauer kann ich das nicht sagen, ich habe sie ohne Namensschild in einem Baumarkt erworben, wo sie mich mit dem typischen "Hol-mich-hier-raus"-Blick unglücklicher Baumarktpflanzen angeschaut hat. Die Orchidee soll andeuten, dass die hier besprochenen Bücher, jedes auf seine Weise, etwas Besonderes sind, so wie die Blüten einer Orchidee.