WI>Data

— Eine Einführung in die Wirtschaftsinformatik

auf der Basis der Web-Technologie —

Hinrich E. G. Bonin[1]

5-Oct-1997 – 06-Oct-2004

[1]Prof. Dr. rer. publ. Dipl.-Ing. Dipl.-Wirtsch.-Ing. Hinrich E. G. Bonin, University of Applied Sciences, Fachhochschule Nordostniedersachsen, Volgershall 1, D-21339 Lüneburg, Germany.

Vorspann

Zusammenfassung

Der Mensch hat dreierlei Wege,
klug zu werden:
erstens durch Nachdenken, das ist der edelste,
zweitens durch Nachahmen, das ist der leichteste,
und drittens durch Erfahrung, das ist der bitterste.
Spruch[1] von Konfuzius

Die Disziplin Wirtschaftsinformatik (WI) befaßt sich mit der Konstruktion, dem Einsatz und den Wirkungen von computergestützten Systemen, die für Unternehmen oder Verwaltungen potentiell nützlich sind. Dazu zählen beispielsweise Dispositionssysteme, Planungssysteme, Entscheidungssysteme, Administrationssysteme und Kontrollsysteme; kurz und allgemein formuliert: Informationssysteme. Damit das Informationssystem nicht zum modischen Selbstzweck entartet, sondern tatsächlich Nutzen bringt, bedarf es einer systematischen Analyse und Modellierung der betriebswirtschaftlichen Zusammenhänge und zwar auch über die Grenzen des einzelnen Unternehmens hinaus. Fragen der Systemanalyse und Modellierung von betriebswirtschaftlichen Daten prägen daher die Konstruktion und den Betrieb von Informationssystemen sowohl für einzelne Funktionsbereiche als auch für ganze Branchen.

Informationssystem

Die WI agiert nicht mehr auf der „grünen Wiese“. Beinahe jede betriebswirtschaftlich-orientierte Organisationseinheit setzt vielfältige In-

[1] Quelle für solche Sprüche siehe zum Beispiel ↪ `http://free.pages.at/webnetz/mensch_hat_dreierlei_wege.htm` (online 9-Oct-2003)

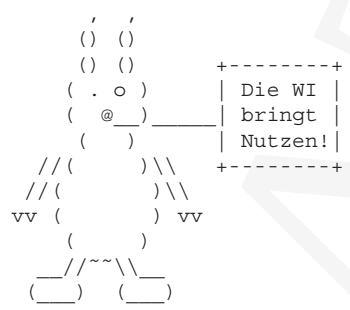
Systemlandschaften

formationssysteme ein und erneuert diese laufend. Eine wichtige WI-Aufgabe ist daher die Pflege und Weiterentwicklung von bestehenden Computerlandschaften. Dazu gehören beispielsweise auch Kriterien zur Bewertung von Standard- oder Branchensoftware sowie Verfahren zur Wirtschaftlichkeitsbeurteilung eines Informationssystems.

Klar ist daher, das WI-Arbeitsfeld ist komplex und stark in Bewegung. Die relevante Hard- und Softwaretechnik ändert sich rasant. Klar ist aber auch, zum Einstieg ist ein Fokus auf ein paar Kernpunkte notwendig. Das vorliegende Buch WI>Data konzentriert sich auf die Handhabung von Daten, das heißt auf die Abbildung von Daten und die Steuerung von Abläufen. WI>Data bedient sich dazu der Technologie, die im World Wide Web (kurz: WWW oder Web) zum Einsatz kommt, das heißt zum Beispiel die Dokumentensprache XML (mit den Ausprägungen HTML, WML, CSS,XSL) und die Programmiersprache Java^{TM}.

Vorwort

Ihr Vorhaben ist gut! Sie möchten sich mit Wirtschaftsinformatik (WI) befassen. Die Disziplin ist stark im Fluß und hoch interessant. Gebraucht werden viele kreative Köpfe, die sich mit allen Aspekten der Hardware, der Software, der betrieblichen Organisation, der Nutzer und der Betroffenen auseinandersetzen.



Auch wenn manch Kritiker meint, es handelt sich um eine Pseudodisziplin — entweder macht man richtig Informatik (↪ Bits&Bytes-Freak) oder richtig Informationsmanagement (↪ Excel-Fuzzy). Gerade das Zusammenspielspiel zwischen Technik einerseits und betriebswirtschaftlicher Organisationen andererseits ist ein äußerst nützliches Arbeitsfeld.

Dieses Buch heisst WI>Data, weil es bei einer Einführung in eine Disziplin darauf ankommt, die Dinge kurz und leicht verständlich, zumindest aber möglichst nicht mißverständlich oder mehrdeutig zu notieren. Aus der Sicht eines Computerprogramms sind Daten „eindeutige“ Bausteine. Es ist das englische Wort *data* gewählt, erstens weil in der WI Anglizismen sehr gebräuchlich (und häufig unvermeidbar) sind und zweitens weil es ein Zeichen kürzer ist als „Daten“ und damit einem Ziel der Informatik, Sachverhalte möglichst kurz zu beschreiben, näher kommt. Das Größerzeichen aus der Mathematik steht zwischen `WI` und `Data` um eine Abbildung zu verdeutlichen. Es symbolisiert, daß es um eine Transformation des relevanten WI-Stoffes auf eindeutige Aussagen, also auf Daten, geht.

WI>Data wurde für diejenigen geschrieben, die mit dem Begriff Computer nicht nur Bildschirm und Tastatur verbinden wollen. WI ist weit mehr als der nützliche Einsatz von Personalcomputern (*Clients*), Workstations (*Servern*), mobilen Telefonen (Handys), Fernsehern und Großleinwänden in einem globalen Netzwerk (Internet). Das intensive Befassen mit der Disziplin WI erzeugt ein eigenständiges Denkmodell, prägt eine persönliche Sicht auf eine „Zusammenarbeit“ zwischen Mensch und Maschine. WI>Data ist daher sicherlich auch ein Ergebnis

der eigenen „Denke“, Erfahrungen und Publikationen. Zu nennen sind hier insbesondere *The Joy of Computer Science* [Bo93], *Arbeitstechniken für die Softwareentwicklung* [Bo92a], *Die Planung komplexer Vorhaben der Verwaltungsautomation* [Bo88], *Software-Konstruktion mit LISP* [Bo91b], *<HTML>-Ratgeber — Multimediadokumente im World-Wide Web programmieren* [Bo96], *Der Java-Coach* [Bo98], *Der kleine XMLer —XML verstehen und anwenden* [Bo00] und *Aspect-Oriented Software Development — A Little Guidence to Better Java Applications* [Bo02].

Wer eine Mensch-Maschine-Kooperation gestaltet will, braucht eine geeignete Vorstellung von der Leistungsfähigkeit und den Restriktionen vernetzter Computer. Dazu ist die Kenntnis der Abarbeitung von Programmen und der Speicherung von Daten erforderlich. WI>Data skizziert diese Arbeitsweise mit Hilfe der Programmiersprache Java^{TM}.

HTML, XML, UML

Zur Notation von Datenstrukturen dienen HTML (Hypertext Markup Language) und XML (Extensible Markup Language). Systemanforderungen und Entwurfsmodelle sind in UML (Unified Modeling Language) dargestellt. Beispielsweise läßt sich der Inhalt von WI>Data dann wie folgt beschreiben:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<WIDATA>
  <CHAPTER>
    (Denk-)Welt der WI
  </CHAPTER>
  <CHAPTER>
    Abbildung von Daten
  </CHAPTER>
  <CHAPTER>
    Rechnen: Steuerung von Abläufen
  </CHAPTER>
  <CHAPTER>
    Softwarekonstruktion:
      Vom Modell zum Produkt
  </CHAPTER>
  <APPENDIX>
    Musterlösungen, Tabellen,
    Abkürzungen, Quellen und Index
  </APPENDIX>
</WIDATA>
<!-- End of object wiinhalt.xml -->
```

Für Ihr Vorhaben können Sie nur ein recht begrenztes Zeitkontingent investieren. Sie müssen und wollen möglichst effektiv Ihre tragfähige WI-Basis schaffen. WI>Data ist daher als ein *Arbeitsbuch zum Selbststudium und für Lehrveranstaltungen* konzipiert. Der WI-Neuling möge es Abschnitt für Abschnitt durcharbeiten. Der WI-Vorbelastete kann zunächst die Zusammenfassungen der einzelnen Abschnitte studieren und dann entscheiden, ob der Inhalt des jeweiligen Abschnittes schon bekannt ist. Dazu ist sicherlich die Bearbeitung der Übungsaufgaben, die am Schluß jedes Kapitels stehen, hilfreich. Damit WI>Data auch als Nachschlagewerk benutzbar ist, enthält es sowohl Vorwärts- wie Rückwärts-Verweise und einen umfassenden Index.

Während der Arbeit am Manuskript lernt man erfreulicherweise stets dazu. Das hat jedoch auch den Nachteil, daß man laufend neue Unzulänglichkeiten am Manuskript erkennt. Schließlich ist es trotz solcher Schwächen der Öffentlichkeit zu übergeben. Ich bitte Sie daher im voraus um Verständnis für Unzulänglichkeiten. Willkommen sind Ihre konstruktiven Vorschläge, um die Unzulänglichkeiten Schritt für Schritt weiter zu verringern. Ihre Vorschläge werden mit Ihrer Zustimmung über den Web-Server:

`http://as.fhnon.de`

verfügbar gemacht. Dort finden Sie auch alle aktuellen Ergänzungen.

Danksagung

Für das Interesse und die Durchsicht einer vorhergehenden Fassung danke ich meinem Kollegen Prof. Dr. Fevzi Belli (Universität Paderborn). Ohne die kritischen Diskussionen mit Studierenden im Rahmen der Lehrveranstaltungen *Einführung in die Wirtschaftsinformatik* im Fachbereich Wirtschaft der Fachhochschule Nordostniedersachsen (FH NON) wäre WI>Data nicht in dieser Form entstanden. Ihnen möchte ich an dieser Stelle ganz besonders danken.

Lüneburg, 6. Oktober 2004
<ERFASSER><VERFASSER>Hinrich E. G. Bonin</VERFASSER></ERFASSER>

Notation

Write once, debug everywhere.
Java^{TM} Reputation
aufgrund der Unterschiede
in den marktüblichen Browsern

In WI>Data wird auf einen Versuch geschlechtsneutrale Formulierungen zu erreichen verzichtet, denn alle bisherigen Lösungsansätze erscheinen unbefriedigend.[2] Aus Lesbarkeitsgründen sind daher nur die männlichen Formulierungen genannt; die Leserinnen seien implizit berücksichtigt. So steht das Wort „Programmierer“ hier für Programmiererin und Programmierer.

Java^{TM}

Für die Notation von Algorithmen wird die Syntax der Programmiersprache **Java**^{TM}[3] verwendet. Diese entspricht weitgehend der Syntax der Sprachen C++ beziehungsweise C. Eine solche Syntax auf der Basis von geschweiften und runden Klammern ist heute weit verbreitet und gehört zum Rüstzeug eines Informatikers.

HTML
XML

Zur Beschreibung der Strukturen in Dokumenten werden HTML[4]- und XML[5]-Konstrukte verwendet. Sind Anforderungen an ein Informationssystem und Entwurfsmodelle zu notieren, dann kommen UML[6]-Konstrukte zum Einsatz.

UML

Ein Programm (Quellcode) ist in der Schriftart `Typewriter` dargestellt. Die ausgewiesenen Zeilennummern in einer solchen Programmdarstellung sind kein Bestandteil des Quellcodes. Sie dienen zur Vereinfachung der Erläuterung.

Ist ein Symbol (oder ein Name) zu erläutern, dann ist zu verdeutlichen, wann der eigentliche Erläuterungstext beginnt und das Symbol endet. Für diese Unterscheidung wird als Metazeichen „≡“ verwendet. Es dient als Beginnmarkierung des Erläuterungstextes. Hier sind zwei Beispiele:

---

[2] Unzweifelhaft bedarf dieses Problem jedoch einer Lösung.

[3] Sun *Java Virtual Maschine* Version 1.2 (auch als Version 2 bezeichnet)

[4] HTML ≡ Hypertext Markup Language in der Definition von XHTML

[5] XML Version 1.0 ≡ Extensible Markup Language — Akronym hätte normalerweise EML (Extensible Markup Language) lauten müssen, aber XML „sounds cooler“.

[6] UML Version 1.4 ≡ Unified Modeling Language

$\hookrightarrow$x $\equiv$ Schauen Sie bei x nach! $\equiv$

APPLIKATION $\equiv$ Anwendungsprogramm einschließlich der Daten

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

(Denk-)Welt der WI

Was ist WI? Eine besondere Informatik oder ein wesentlicher Teil einer ganzheitlichen Informatik? In jedem Fall „übernimmt“ die WI von der Informatik das Spannungsfeld zwischen deren Wurzeln Mathematik und Elektrotechnik einerseits und ihrer Leitbildaufgabe für soziale Organisationen (Staat, Unternehmen, Behörden, Familien usw.) andererseits. Betriebliche Arbeitsprozesse neu zu gestalten bedeutet auch Ansichten zu verändern und Verhaltensweisen aufeinander abzustimmen.

In diesem Kapitel werden Hardware, Software, Organisation, Nutzer und Betroffene als wesentliche Elemente der WI betrachtet. Im Mittelpunkt steht dabei die Perspektive der Übertragung von Arbeit auf Maschinen.

Wegweiser

Das Kapitel „(Denk-)Welt der WI“ erläutert:

- die Einordnung der WI als eigenständige Disziplin,
  ↪ Seite 22 . . .

- die Entwicklung vom Einprogrammsystem zur weltweiten Infrastruktur bestehend aus einem Konglomerat unterschiedlichster Informationstechnik und
  ↪ Seite 40 . . .

- die Begriffe Hardware, Software, Organisation, Benutzer & Betroffene als prägende Elemente der Wirtschaftsinformatk.
↪ Seite 51 . . .

1.1 Was ist oder bedeutet WI?

Wirtschaftsinformatiker beschäftigen sich mit Gestaltung und Betrieb von Systemen der computergestützten Informationsverarbeitung für betriebswirtschaftliche Aufgaben.
↪[Me+04]

WI=?

Was ist WI? Was bedeutet der Begriff WI? Wie kann man WI definieren — in der Vergangenheit, in der Gegenwart und in der Zukunft? Diese Art von „Was-ist-Frage“ ist schnell nur noch interessant für Dispute im Kreis von Philosophen (in ihrem Elfenbeinturm?). Schnell geht es dabei hauptsächlich um feinste Feinheiten der Begriffsdefinitionen. Wenn wir annehmen: Informatik sei ≈ eine Wissenschaft, die sich primar mit Informationen im Kontext von Maschinen befasst („Maschinisierung von Kopfarbeit“↪ [Na92]), dann stellt sich die Frage nach der Begriffsdefinition „Information“. Informatiker stützen ihre Definition auf zwei Punkte (↪ [Rech04] S. 92):

Information

1. Information ist gedeutete (interpretierte) Nachricht oder Mitteilung. Sie entsteht erst beim Empfänger der Nachricht.

2. Information an sich, als reale Erscheinung, die für sich selbst existiert, die transportiert und womöglich gemessen, in Teile zerlegt oder aus Teilen zusammengesetzt werden kann, gibt es nicht. Der Begriff „Information“ ist vielmehr eine Verdinglichung des Informierens (*Hypostasierung*, wie die Philosophen sagen), so wie viele abstrakte Substantive Hypostasierung sind, zum Beispiel„Sein“, „Nichts“, „Identität“, „Denken“.

Üblich ist auch eine Unterscheidung in syntaktische und semantische Information (↪ [Rech03] S. 321):

Syntax

- Syntaktische Information

  - ist ein Maß für die kürzeste Codierung der Nachricht
  - ist quantifizierbar
  - braucht keinen Empfänger, steckt objektiv in der Nachricht
  - ist bestimmt durch Alphabet und Auftrittswahrscheinlichkeit seiner Zeichen

- Semantische Information **Semantik**
  - bezeichnet die Bedeutung einer Nachricht für den Empfänger
  - ist nicht quantifizierbar
  - braucht einen Empfänger und entsteht erst bei ihm
  - bestimmt durch die Bedeutung für den Empfänger

Allerdings gilt noch immer die These: *Das Informationszeitalter kann sich nicht einigen über den Begriff „Information“*. (↪[Kl03] S. 267)[1]

Um solche, mehr oder wenig praxisrelevante Dispute einzugrenzen, wird in WI>Data die notwendige Selbstverständnisfrage genutzt, um gleichzeitig Möglichkeiten und Grenzen einer formalen Notation aufzuzeigen. Eine Antwort auf die WI-Selbstverständnisfrage wird deutlich anhand der Erläuterung von charakteristischen Elementen und durch eine Beschreibung des Verhältnisses zur Informatik einerseits und zur Betriebswirtschaft andererseits.

1.1.1 Informatik⇔WI-Beziehung — in UML

Eine zentrale Frage zum Sebstverständnis der WI lautet: Ist die WI eine selbständige *Angewandte Informatik* oder eine besondere *Anwendung der Informatik*?

Die erste Auffasung entspricht eher einer Aggregationsbeziehung zwischen Informatik und WI. Die WI ist dann gleichzeitig (Bestand-)-Teil der Disziplin der Informatik und der Betriebswirtschaft (↪Bild 1.2 Seite 25).

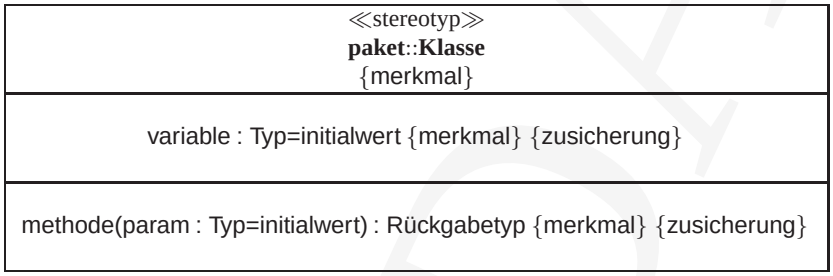
Die zweite Auffassung betont eher eine Vererbungsbeziehung. Die WI ist dann eine spezielle Ausrichtung der Disziplinen Informatik und

[1] Das Diskussionsfeld ist beim Begriff „Information“ sehr weit und reicht tief in die Vergangenheit zurück. So wird beispielsweise Moses von lateinischen Autoren *informator populi* genannt und bei Thomas von Aquin heißt Bildung „informatio“. (↪[Kl03] S. 268)

Klassensymbol:

Eine Klasse beschreibt Zustände, Eigenschaften, Merkmale und/oder Verhältnisse von (vielen) ähnlichen Objekten, die auch als ihre Exemplare oder ihre Instanzen bezeichnet werden. Eine Klasse stellt die Gemeinsamkeiten dieser Objekte dar. Dazu beschreibt sie mögliche Werte in Form von Variablen und deren Darstellung und Änderung in Form von Methoden.

Klassensymbol mit Variablen und Methoden:



Legende:

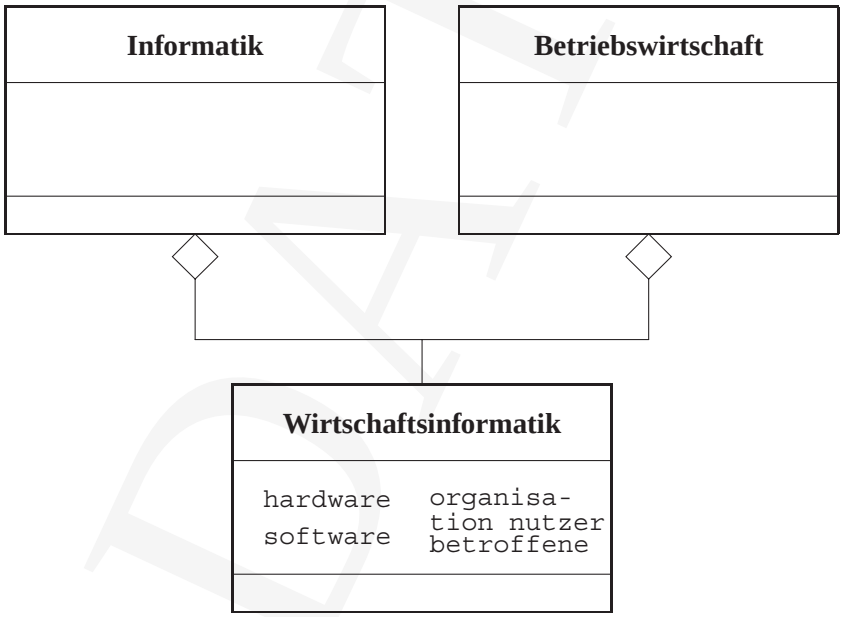
≪stereotyp≫	≡	Bezeichnung einer besonderen Einteilung z. B. ≪fachklasse≫
paket	≡	Einheit für mehrere Klassen

Abbildung 1.1: UML-Basiselement: Klasse

Betriebswirtschaft. Beide sind in der Elternrolle und übertragen auf die WI ihr gesamtes Wissen (↪Bild 1.3 Seite 26).

Diese unterschiedlichen Zuordnungen der WI zu den Disziplinen Informatik und Betriebswirtschaft lassen sich durch eine formale Notation betonen. In beiden Bildern ist die WI als Klasse notiert und zwar gemäß der *Unified Modeling Language* (UML).[2] Zur Erläuterung der Klassendarstellung in UML ↪Bild 1.1 auf Seite 24.

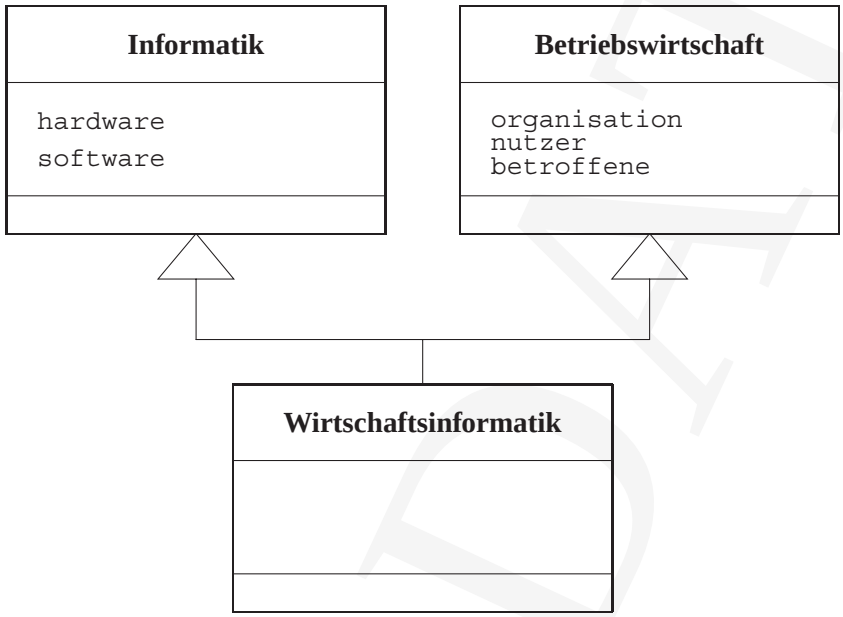
[2] Eine kurze UML-Einführung enthält zum Beispiel [Fo+98]. Für eine umfassendere Darstellung ↪[Neu98].



Legende:

X	≡	Klasse X, Symbolerläuterung ↪Bild 1.1 Seite 24
X <> Y	≡	Aggregation: Y ist Bestandteil von X
name	≡	Attribut, auch Slot oder Variable genannt

Abbildung 1.2: WI als Bestandteil der Informatik — in UML



Legende:

X	≡	Klasse X, Symbolerläuterung ↪Bild 1.1 Seite 24
X ◁— Y	≡	Vererbungsbeziehung: Y hat die Eigenschaften von X
name	≡	Attribut, auch Slot oder Variable genannt

Abbildung 1.3: WI als spezielle Informatik — in UML

1.1.2 Elemente der WI — in XML

Im Regelfall sind griffige Definitionen für die Praxis nicht hinreichend und darüberhinaus vielfätig interpretierbar und daher mißverständlich. Dies gilt sicherlich auch für die Aussage:

> „WI ist diejenige Informatik, die für Unternehmen und Verwaltungen nützlich ist.“

WI>Data macht es sich daher einfach und verzichten auf einen (langen) textliche Disput. Hier wird davon ausgegenagen, daß es unstrittig ist, daß der Gegenstand der Disziplin WI mindestens die folgenden, fünf wesentlichen Elemente umfaßt:

Elemente

- Hardware
- Software
- Organisation
- Nutzer
- Betroffene

Im konkreten Arbeitsbereich der WI haben diese eng miteinander verzahnten Elemente ganz unterschiedliche Ausprägungen. Solche unterschiedlichen Ausprägungen beeinflussen zwangsläufig das individuelle Bild von der WI-Disziplin. Zum Beispiel kann die WI-(Denk)Welt primär geprägt sein von den persönlichen Erfahrungen mit dem Umgang eines Notebooks oder mit einem voll computerisierten Unternehmen. Ein verantwortlicher Konzernmanager wird beispielsweise seine Erfahrungen mit großen Rechenzentren und mit vielen hunderten Computern im Netz in den Mittelpunkt seiner WI-Betrachtung stellen. Beide Sichten, Erklärungsmuster, Erwartungen, Einschätzungen; kurz WI-Denkwelten, sind sehr unterschiedlich.

In WI>Data wird diese Spannweite individueller WI-Denkwelten auf der Basis der fünf Elemente jetzt formaler notiert. Dazu wird XML[3]

[3] Es gibt zu XML eine Menge Publikationen, beispielsweise auch eine von mir ↪[Bo00]. Manche beziehen sich auf einen Browser. So zum Beispiel ↪[Ho99]. Eine gelungene XML-Erläuterung ist beispielsweise ↪[DuCh1999]. XML im Kontext seiner Entwicklung und seiner Alternativen erläutert beispielsweise ↪[Go+99].

genutzt. Die *Extensible Markup Language*, primär konzipiert zum Auszeichen von Web-Dokumenten, ist formal gesehen eine Meta-Grammatik für kontextfreie Grammatiken; also ein „Instrument“ (eine Grammatik) mit der sich Grammatikregeln[4] notieren lassen.

Um XML schon an dieser Stelle verwenden zu können, sind vorab ein paar Einstiegshinweise erforderlich. Besteht ein Baustein `X`, in XML als Element bezeichnet, aus den Elementen `A`, `B` und `C`, wobei es sich jeweils um Text handeln möge, dann läßt sich notieren:

XML

```
<!ELEMENT X (A, B, C)>
<!ELEMENT A (#PCDATA)>
<!ELEMENT B (#PCDATA)>
<!ELEMENT C (#PCDATA)>
```

`#PCDATA` Die Angabe `#PCDATA` steht für *Parsed Character Data* und bedeutet, daß der Inhalt des Elementes nur aus vorgegebenen Textzeichen besteht. Die Zeichen können unterschiedlich codiert sein. Das *Character Encoding*[5] wird wie folgt notiert:

- 8-Bit-UTF-Code („7-Bit ASCII-Code“):
  `<?xml encoding="UTF-8"?>`
- 16-Bit UTF-Code[6]:
  `<?xml encoding="UTF-16"?>`
- Latin 1 („Europa-Code“):
  `<?xml encoding="ISO-8859-1"?>`

Latin 1 Ohne explizite Codierungsangabe wird `UTF-8` angenommen. Für Texte in Deutsch ist jedoch der Code „Latin 1“ bedeutsam, um die Umlaute mit einem Zeichen darstellen zu können.

Soll das Element `X` nicht zwingend aus einem Element `A`, einem Element `B` und einem Element `C` bestehen, wie bisher notiert, dann sind folgende Zusatzangaben zur **Multiplizität** möglich:

---

[4] In der XML-Welt spricht man von einer *Document Type Definition* (DTD) oder von einem Schema anstatt von einer Grammatik.

[5] Näheres zur Codierung ↪Abschnitt 2.1.1.

[6] UCS Transformation Format 16; spezifiziert in *ISO 10646 character encodings*. UCS ≡ *Universal Multiple-Octet Coded Character Set*. Ein XML-Processor muß mindestens `UTF-8` und `UTF-16` unterstützen.

? Das Fragezeichen gibt an, daß ein Element fehlen oder genau einmal vorkommen kann.

Multiplizität

∗ Der Stern gibt an, daß ein Element fehlen oder auch mehrmals vorkommen kann.

\+ Das Pluszeichen gibt an, daß ein Element einmal oder mehrmals vorkommen kann.

| Der senkrechte Strich trennt Elemente in einer Aufzählung, wenn jeweils eines der aufgezählten Elemente vorkommen kann.

Besteht `X` zum Beispiel stets aus einem oder mehreren Elementen `A` , aus einem Element `B` und einem oder keinem Element `C`, dann wird diese Aussage wie folgt notiert:

```
<!ELEMENT X (A+, B, C?)>
```

DTD

Mit diesem XML-Syntaxwissen ist man nun in der Lage Hardware, Software, Organisation, Nutzer und Betroffene als Elemente der WI zu definieren. Die sogenannte *Document Type Definition* (DTD) steht in der Datei `wi.dtd`. In der Datei `wisingle.xml` ist die Welt des einzelnen Notebooknutzers beschrieben. Das Bild 1.4 auf Seite 31 zeigt ihre Darstellung im Browser[7]. Die Datei `wiglobal.xml` beschreibt das Szenario eines Konzernmangagers. Ihre Browser-Darstellung zeigt Bild 1.5 auf Seite 32.

`wi.dtd`

```
 1  <?xml version="1.0" encoding="ISO-8859-1"?>
 2  <!-- Bonin    01-Sep-1999                -->
 3  <!ELEMENT WI (HARDWARE, SOFTWARE,
 4    ORGANISATION, NUTZER, BETROFFENE)>
 5  <!ELEMENT HARDWARE (#PCDATA)>
 6  <!ELEMENT SOFTWARE (#PCDATA)>
 7  <!ELEMENT ORGANISATION (#PCDATA)>
 8  <!ELEMENT NUTZER (#PCDATA)>
 9  <!ELEMENT BETROFFENE (#PCDATA)>
10  <!-- End of object wi.dtd               -->
11
```

[7] Microsoft Internet Explorer 5 Version: 5.00.2014.0216

wisingle.xml

```
 1  <?xml version="1.0" encoding="ISO-8859-1"?>
 2  <!-- Bonin 01-Sep-1999                    -->
 3  <!DOCTYPE WI SYSTEM "wi.dtd">
 4  <WI>
 5    <HARDWARE>
 6      Notebook
 7    </HARDWARE>
 8    <SOFTWARE>
 9      MS Windows NT 4.0,
10      MS Office
11    </SOFTWARE>
12    <ORGANISATION>
13      Handwerksbetrieb
14    </ORGANISATION>
15    <NUTZER>
16      DV-Freak
17    </NUTZER>
18    <BETROFFENE>
19      Kollegen, Familie
20    </BETROFFENE>
21  </WI>
22  <!-- End of object wisingle.xml        -->
23
```

Die Verknüpfung dieser Datei zu der obigen *Document Type Definition* (DTD) erfolgt über das Statement in Zeile 3. Dieses Statement bezeichnet man als *Document Type Declaration*. Es gibt für das Ausgangselement `WI` (*root node*) an, daß seine DTD über das System zu laden ist.[8]

[8] Die DTD für das Element `WI` ist also extern gespeichert und nicht in dem Dokument selbst enthalten. Eine interne DTD wäre eingerahmt von eckigen Klammern „`[...]`“ zu notieren; in unserem Beispiel folgendermaßen:

```
<!DOCTYPE WI [
  <!ELEMENT WI (HARDWARE, SOFTWARE,
    ORGANISATION, NUTZER, BETROFFENE)>
  <!-- Rest der DTD -->
]>
<WI>
  <HARDWARE>
    Notebook
  </HARDWARE>
...
```

Zu beachten ist, daß der *Root-Element*-Name mit dem *Root-Element* in der Deklaration

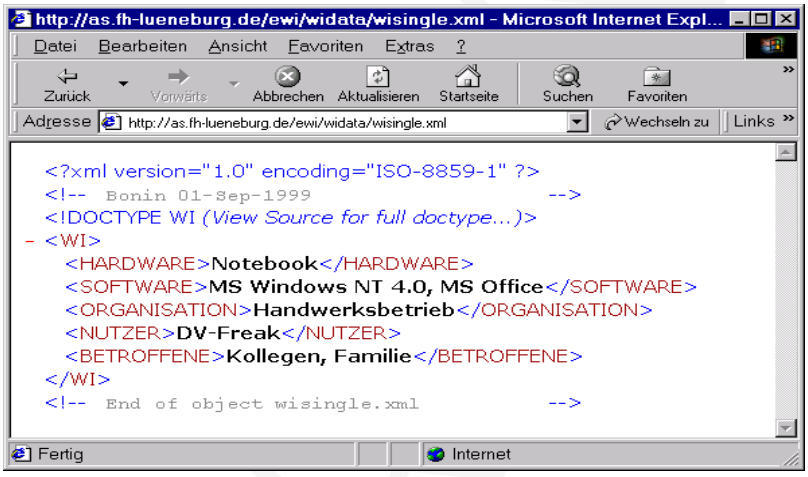


Abbildung 1.4: Elementebeispiel für die WI-Welt eines Einzelkämpfers — in XML

wiglobal.xml

```
 1  <?xml version="1.0" encoding="ISO-8859-1"?>
 2  <!-- Bonin 01-Sep-1999                    -->
 3  <!DOCTYPE WI SYSTEM "wi.dtd">
 4  <WI>
 5    <HARDWARE>
 6      Client/Server-Netzwerk auf Basis von
 7      Intel-PCs und SUN-Workstations
 8    </HARDWARE>
 9    <SOFTWARE>
10      alle Windows- und
11      verschiedene UNIX-Betriebssysteme,
12      verschiedene Kommunikationsprotokolle,
13        zum Beispiel HTTP1.1,
14      betriebswirtschaftliche Standardpakete,
15        zum Beispiel SAP R3 und Baan,
16      verschiedene Middleware- und Web-Server,
17        zum Beispiel mit Servlets und RMI
```

übereinstimmt.

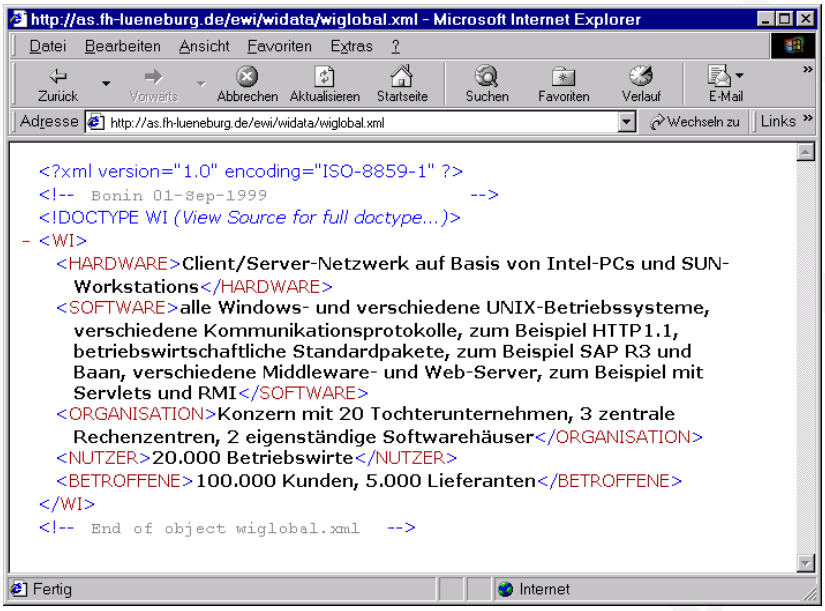


Abbildung 1.5: Elementebeispiel für die WI-Welt eines Konzernmanagers — in XML

```
18    </SOFTWARE>
19    <ORGANISATION>
20      Konzern mit 20 Tochterunternehmen,
21      3 zentrale Rechenzentren,
22      2 eigenständige Softwarehäuser
23    </ORGANISATION>
24    <NUTZER>
25      20.000 Betriebswirte
26    </NUTZER>
27    <BETROFFENE>
28      100.000 Kunden,
29      5.000 Lieferanten
30    </BETROFFENE>
31  </WI>
32  <!-- End of object wiglobal.xml -->
33
```

Man kann auch die Auffassung vertreten, die Disziplin WI „bestehe“ nicht aus den genannten Elementen, sondern die „Elemente“ haben eher

den Charakter von Eigenschaften. Im Sinne der XML-Notation handelt es sich dann um Attribute zu einem Element WI. Die fünf Attribute werden zuammengefaßt in einer Liste. Eine solche Attributliste[9] hat in XML folgende Struktur: ATTLIST

```
<!ATTLIST element-name
    attribut-name attribut-daten angabe-notwendigkeit
    ... gegebenenfalls weitere attribute ...>
```

Man kann dann die WI-Beschreibung auf der Meta-Ebene, also der *Document Type Definition*, wie in Datei `wiattri.dtd` (↪Seite 33) notieren. Das Bild 1.6 auf Seite 34 zeigt die WI-Darstellung mittels Attribute bezogen auf der (Denk-)Welt des Einzelkämpfers. Dabei nutzt der Browser die XML-Möglichkeit leere Konstrukte mit einer Marke zu notieren. Dazu wird die Endmarke als Schrägstrich in der Anfangsmarke abgebildet; also statt `<WI ...></WI>` wird `<WI .../>` angezeigt.

wiattri.dtd

```
 1  <?xml version="1.0" encoding="ISO-8859-1"?>
 2  <!-- Bonin    12-Oct-1999               -->
 3  <!ELEMENT WI EMPTY>
 4  <!ATTLIST WI
 5    HARDWARE CDATA #REQUIRED
 6    SOFTWARE CDATA #REQUIRED
 7    ORGANISATION CDATA #REQUIRED
 8    NUTZER CDATA #REQUIRED
 9    BETROFFENE CDATA #REQUIRED>
10  <!-- End of object wiattri.dtd              -->
11
```

wiattris.xml

```
 1  <?xml version="1.0" encoding="ISO-8859-1"?>
 2  <!-- Bonin 12-Oct-1999                  -->
 3  <!DOCTYPE WI SYSTEM "wiattri.dtd">
 4  <WI
 5   HARDWARE="Notebook"
 6   SOFTWARE="MS Windows NT 4.0, MS Office"
 7   ORGANISATION="Handwerksbetrieb"
```

[9] Näheres zur Attributen ↪`booklist.dtd` Seite 96.

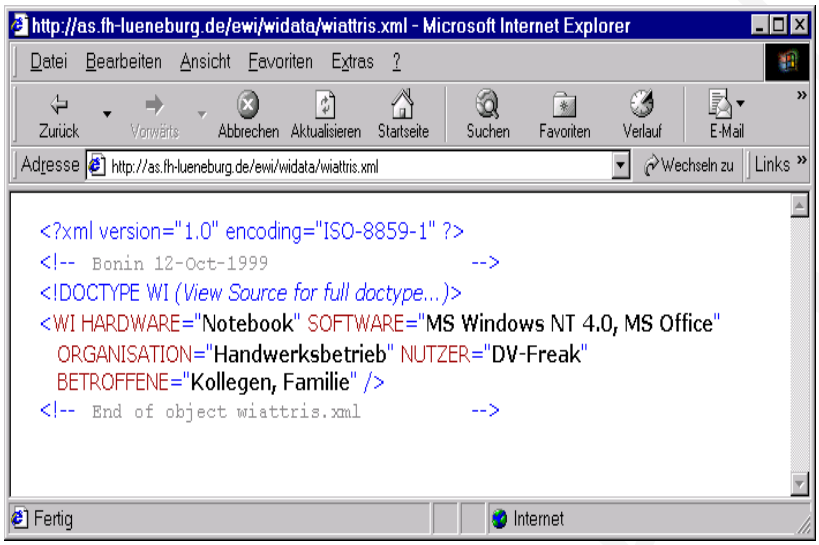


Abbildung 1.6: Attribute-Beispiel für die WI-Welt eines Einzelkämpfers — in XML

```
 8   NUTZER="DV-Freak"
 9   BETROFFENE="Kollegen, Familie">
10  </WI>
11  <!-- End of object wiattris.xml        -->
12
```

1.1.3 Objekt-orientierte WI-Sicht — in UML und Java

In der Informatik und damit auch in der WI[10] wird die „reale Welt“ oft als eine Menge von kommunizierenden Objekten betrachtet. Aus dieser objekt-orientierten Perspektive[11] lassen sich die skizzierten Elemente der WI auch als Objekte notieren. In WI>Data wird für diese objekt-orientierte Sicht als Darstellungsmittel UML (↪Bild 1.7 Seite 35) und die Programmiersprache Java^{TM} verwendet. Der Kern der objekt-orientierten Sicht in Java^{TM} ist die Erzeugung von Objekten (≡ Instanzen[12]), die miteinander kommunizieren können, aus Klassen. Ei-

Java^{TM} Klasse

[10] mindestens bei der „Vererbungssicht“ ↪Seite 26

[11] Alternative Perspektiven sind beispielsweise die funktionale Sicht oder die imperative Sicht.

[12] auch *Member* genannt.

Legende:

+	≡	allgemein zugreifbar	— in Java: `public`
-	≡	nicht allgemein zugreifbar	— in Java: `private`
`methode()`	≡	Klassenmethode	— in Java: `static`

Abbildung 1.7: Objekt-orientiertes Beispiel der WI-Denkwelt — in UML

ne Klasse ist eine Beschreibung der Merkmale, Zustände, Eigenschaften und Verhaltensweisen ihrer Objekte; im folgenden Instanzen genannt. Die Klassendefinition entspricht daher der DTD wobei die Klassendefinition zusätzlich die Operationen definiert, die auf ihre Instanzen ausgeführt werden können. Diesen „aktivierbaren“ Teil der Klasse bezeichnet man als Methodenteil, während der „passive“ Teil aus den Beschreibungen der Variablen besteht. Die Methode mit dem selben Namen wie die Klasse dient zur Erzeugung einer Instanz. Man nennt sie auch Konstruktor. Ein Standardkonstruktor wird in Java^{TM} für jede Klasse automatisch erzeugt. Die Kommunikation zwischen den Objekten läßt sich als Nachrichtenaustausch[13] interpretieren. Ein Objekt empfängt eine Nachricht und gibt selbst eine Antwort an den Sender zurück. Der Nachrichtenempfang wird als Aufruf einer Methode abgebildet, wobei der jeweilige „Inhalt“ als Parameterwert mitgegeben wird. Die Antwort ist der Rückgabewert der Methode. Die Erzeugung eines Objektes und den Methodenaufruf notiert man folgendermaßen:

Nachricht

Klassendefinition `zugriffsrecht class MyKlassenName{variablen; methoden();}`

Variablendefinition `zugriffsrecht Typ myVariablenName;`

Methodendefinition `zugriffsrecht TypRückgabeWert myMethodenName(ParameterTyp parameter);`

Instanzerzeugung `Typ myObjektName = new Konstruktor();`
Hinweis: `MyKlassenName`, `Konstruktor()`, und `Typ` sind hier gleichnamig.

Methodenaufruf `myObjekt.myMethodenName(parameter);`
Hinweis: Zwischen dem Objekt und der Methode steht ein **Punkt**. Dieser dient zur Abgrenzung des Objektnamens vom Methodennamen.

Die Elemente der WI, die bisher als *Document Type Defintion* in der Datei `wi.dtd` (↪Seite 29) standen, können in der Java-Klasse

[13] Der klassische Vertreter für das Nachrichtenkonzept ist die Programmiersprache Smalltalk. Bei ihr wird selbst die Addition von „$1 + 2$“ als Nachrichtenversand aufgefaßt. Die Zahl 1 erhält die Nachricht „$+$“ mit dem Inhalt 2.

`WI.java` (↪Quellcode 21 auf Seite 38) als Instanzvaribalen[14] abgebildet werden. In diesen Instanzvariablen wird wieder die Beschreibung als reiner Text aufgehoben. Dazu wird der Datentyp `String`, also die Form einer Zeichenkette, genutzt. Da in objekt-orientierter Software der Zugriff auf die Internas eines Objektes für alle Zugreifer verborgen wird, werden die Instanzvariablen als `private` definiert. Zum Zugriff wird dann für jede Instanzvariable eine eigene Zugriffsmethode, die öffentlich (`public`) ist, definiert.

„Getter“

Bei den Zugriffsmethoden wird unterschieden, ob sie (nur) den Wert der Variablen zurückliefern (≡ get-Methoden; kurz: „Getter“) oder den Wert der Variablen ändern (≡ set-Methode; kurz: „Setter“). Für das Ausgeben einer Zeile auf dem Bildschirm wird die Konstruktion:

„Setter“

`System.out.println(`String parameter`)`

genutzt. Dabei wird die Methode `println()` auf das Objekt angewendet, das die Klasse `System` unter ihrer Klassenvariablen `out` speichert. Hier ist bedeutsam, daß der Parameter eine Zeichenkette ist und zwar die Zeichenkette, die ausgegeben wird. Eine Zeichenkette (String) läßt sich aus zwei Zeichenketten zusammensetzten indem zwischen den beiden Teilketten ein Pluszeichen „+“ gesetzt wird. Ist beispielsweise die Zeichenkette `"Alles klar?"` auf dem Bildschirm auszugeben, dann kann dies mit folgender Java-Zeile geschehen:

```
System.out.println("Alles " + "klar?");
```

Die WI-Denkwelt, die in der Datei `wisingle.xml` (↪Seite 30) beschrieben wurde, läßt sich nun in einer Java-Klasse `WIBeispiele.java`[15] (↪Quellcode 21 auf Seite 39) notieren. In dieser Klasse ist auch eine primitive Ausgabe programmiert. Um diese Ausgabe auf dem Bildschirm zu erhalten bedarf es erst einer Übersetzung der Java-Klasse in den Byte-Code, der dann von der *Java Virtual Maschine* ausgeführt werden kann. Die Protokolldatei `WIBeispiele.log` (↪ Seite 40) zeigt die Übersetzung mit Java-Compiler `javac` und die Ausgabe der `wisingle`-Werte.

[14] Eine Instanzvariable wird auch Attribut oder Slot genannt.

[15] Um das Quellcodelisting nicht zu lang werden zu lassen, ist hier nur das WI-Objekt `wisingle` vollständig notiert. Für das WI-Objekt `wiglobal` gilt Analoges.

WI.java

```
 1  /**
 2   *  WI-Denkwelt:  Elemente
 3   * @author Hinrih Bonin
 4   * @version 1.0
 5   * @since 11-Nov-1999
 6   */
 7
 8  public class WI {
 9
10    // Elemente
11    private String hardware;
12    private String software;
13    private String organisation;
14    private String nutzer;
15    private String betroffene;
16
17    // Getter
18    public String getHardware() {
19      return hardware; }
20    public String getSoftware() {
21      return software; }
22    public String getOrganisation() {
23      return organisation; }
24    public String getNutzer() {
25      return nutzer; }
26    public String getBetroffene() {
27      return betroffene; }
28
29    // Setter
30    public void setHardware(String h) {
31      hardware = h; }
32    public void setSoftware(String s) {
33      software = s; }
34    public void setOrganisation(String o) {
35      organisation = o; }
36    public void setNutzer(String n) {
37      nutzer = n; }
38    public void setBetroffene(String b) {
39      betroffene = b;}
40  }
41
```

WIBeispiele.java

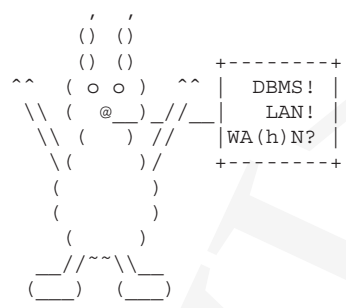
```
 1  /**
 2   * Beispiele für WI-Denkwelten
 3   * @author Hinrih Bonin
 4   * @version 1.0
 5   * @since 11-Nov-1999
 6   */
 7
 8  public class WIBeispiele {
 9
10    public static void main(String argv[]) {
11
12      // Erzeugen von Instanzen der Klasse WI
13      WI wisingle = new WI();
14      WI wiglobal = new WI();
15
16      // Setzen der WI-Elementwerte
17      wisingle.setHardware("Notebook");
18      wisingle.setSoftware("MS Windows NT 4.0, MS Office");
19      wisingle.setOrganisation("Handwerksbetrieb");
20      wisingle.setNutzer("DV-Freak");
21      wisingle.setBetroffene("Kollegen, Familie");
22
23      // Primitives Ausgeben
24      System.out.println("WI-Denkwelt: wisingle");
25      System.out.println(" Hardware: " +
26        wisingle.getHardware());
27      System.out.println(" Software: " +
28        wisingle.getSoftware());
29      System.out.println(" Organisation: " +
30        wisingle.getOrganisation());
31      System.out.println(" Nutzer: " +
32        wisingle.getNutzer());
33      System.out.println(" Betroffene: " +
34        wisingle.getBetroffene());
35
36      // Hier ist Analoges für wiglobal zu notieren
37    }
38  }
39
```

`WIBeispiele.log`

Hier steht das Größerzeichen „>“ als Prompzeichen. Mit diesem Zeichen wird zu einer Eingabe aufgefordert. Das *Hash*-Zeichen „#“ ist das Kommentarzeichen, das heißt, die Zeichen in seiner Zeilen dienen nur zum Verständnis, haben aber keine Wirkung auf die Ausführung durch den Computer.[16]

```
 1  >#  Compilierung; d.h. Java-Quellcode in Byte-Code
 2  >#  für die JVM übersetzen
 3  >javac WI.java
 4  >javac WIBeispiele.java
 5  >#
 6  >#  Aufruf der Methode main() der Klasse WIBeispiel
 7  >#    durch Aufruf der JVM mit dem Klassennamen.
 8  >java WIBeispiele
 9  WI-Denkwelt: wisingle
10   Hardware: Notebook
11   Software: MS Windows NT 4.0, MS Office
12   Organisation: Handwerksbetrieb
13   Nutzer: DV-Freak
14   Betroffene: Kollegen, Familie
15  >
16
```

1.2 WI-Entwicklung: Einprogrammsystem ⇒ WAN

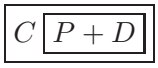


Ein Kennzeichen für die Entwicklung der Informatik im betriebswirtschaftlichen Anwendungsfeld (— früher auch als „kommerzielle Datenverarbeitung“ bezeichnet —) ist die Einsatzform des **Computers** (im folgenden kurz als C notiert), das heißt, die Art und Weise wie Anwendungen gestaltet werden konnten. Eine Anwendung, auch **Applikation** genannt, besteht aus dem **Programm** (P), im Kern den Instruktionen, die die Abarbeitung von C steuern und den eigentlichen **Daten** (D) der Anwendung.

[16] Näheres zur Kommandoabarbeitung ↪Abschnitt 3.2 Seite 159.

Es lassen sich folgende Entwicklungsphasen unterscheiden, wobei heute alle Formen auch in vielfältiger Kombination im betrieblichen Alltag vorkommen.

1.2.1 Einprogrammsystem

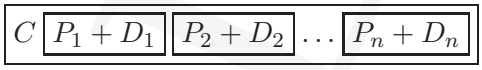


Ein Anwendungsprogramm P wird zusammen mit den benötigten Daten D für einen ganz bestimmten Computer C entworfen und realisert. Die feste Verknüpfung zwischen P und D war in den Anfängen der Datenverarbeitung (DV) deutlich sichtbar — P und D befanden sich in einem Lochkartenkasten oder waren hintereinander auf einem Lochstreifen gestanzt. In XML wird das `EINPROGRAMM-SYSTEM`, bestehend aus `C` und `APPLIKATION`, wobei das Element `APPLIKATION` sich aus `P` und `D` zusammensetzt, wie folgt notiert.

DV-Anfang

```
<!ELEMENT EINPROGRAMM-SYSTEM (C, APPLIKATION)>
<!ELEMENT APPLIKATION (P, D)>
```

1.2.2 Mehrprogrammsystem



In der nächsten Entwicklungsphase stand die Kapazitätssteigerung von C im Mittelpunkt. Es galt die Leerlauf- und Wartezeiten von C zu reduzieren. Dazu wurden Betriebssysteme[17] genutzt, die eine quasi gleichzeitige Abarbeitung von mehreren Programmen auf einem Computer C ermöglichen (*timesharing*). Die feste Verknüpfung von P_i mit allen benötigten Daten D_i ändert sich nicht. In XML wird das Vorkommen von mehreren Elementen `APPLIKATION` durch ein Pluszeichen „+“ dargestellt.

Betriebssystem

```
<!ELEMENT MEHRPROGRAMM-SYSTEM (C, APPLIKATION+)>
<!ELEMENT APPLIKATION (P, D)>
```

[17] Ein Betriebsystem ist eine Sammlung von Programmen, die organisatorische Abläufe und die Kommunikation mit den angeschlossenen Geräten eines Computers steuern. — Beispiel sind *Linux* (Linus Torvald, 1991), die *Windows*-Familie, oder auch *OS/390* (IBM).

1.2.3 Anwendungen auf der Basis eines DBMS

$$\boxed{C\ \boxed{P_1 + D'_1}\ \boxed{P_2 + D'_2}\ \ldots\ \boxed{P_n + D'_n}\ \boxed{DBMS + D_{allgemein}}}$$

DBMS

Mit der Menge der Anwendungsprogramme, die ein Unternehmen im Einsatz hat, wächst auch die Menge der redundanten Daten in $D_{i..n}$. Viele Programme nutzen die gleichen Daten, zum Beispiel die Postadressen der Kunden oder die Verkaufspreise der Produkte. Die Pflege von redundanten Dateien ist arbeitsaufwendig und fehleranfällig. Die von mehreren Programmen genutzten Daten $D_{allgemein}$ werden daher aus der festen Verknüpfung von $P_i + D_i$ entkoppelt und durch ein gemeinsames Datenbankmanagementsystem ($DBMS$) für alle $P_{i..n}$ bereitgestellt. Die Festlegung von $D_{allgemein}$ (Konzeptionierung eines unternehmensweiten Datenmodells) und die Pflege der von vielen Programmen genutzten Datenbank ist eine eigenständige IT-Aufgabe für Experten. Bei der Entwicklung eines Anwendungsprogramms P_i braucht der Programmierer sich „nur“ auf die zusätzlichen, ganz spezifischen Daten D'_i zu konzentrieren. Das $DBMS$ stellt die allgemeinen Daten auf aktuellem Stand bereit. Beispiele für eine leistungsfähige DBMS-Software sind ADABAS[18] (Software AG), DB2[19] (IBM), ORACLE[20] (Oracle Corporation) und POET[21] (POET Software GmbH).

In XML läßt sich die `DATENBANK` bestehend aus den Elementen `DBMS` und `D` abbilden. Das `DBMS` selbst ist ein Programm `P` und benötigt zu seiner Konfiguration Daten `D`.

```
<!ELEMENT DATENBANK-SYSTEM
  (C, APPLIKATION*, DATENBANK)>
<!ELEMENT APPLIKATION (P, D?)>
<!ELEMENT DATENBANK (DBMS, D)>
<!ELEMENT DBMS (P, D)>
```

[18] Näheres zu ADABAS ↪`http://www.softwareag.com/adabas/` Zugriff 22-Nov-1999

[19] Näheres zu DB2 ↪`http://www-4.ibm.com/software/data/db2/index.html` Zugriff 22-Nov-1999

[20] Näheres zu Oracle ↪`http://www.oracle.com/database/` Zugriff 22-Nov-1999

[21] Näheres zu Poet ↪`http://www.poet.de/` Zugriff 22-Nov-1999

Hinweis: Vom einem DBMS ist ein *Content Management System* (CMS) zu unterscheiden. Auch es verwaltet Daten, die für vielfältige Anwendungen benötigt werden. Ein CMS dient jedoch primär zur Erzeugung von Dokumenten, die im Web bereitgestellt werden, siehe beispielsweise die Systeme der Imperia AG[22]

CMS

1.2.4 Übergang zum Präsentationsstandard: Dreischichtensystem

$$C\ \boxed{GUI}\ \boxed{P'_1 + D'_1}\ \boxed{P'_2 + D'_2}\ \ldots\ \boxed{P'_n + D'_n}\ \boxed{DBMS + D_{allgemein}}$$

Die Analyse der Menge der Anwedungsprogramme zeigt, daß sich ein wesentlicher Codeanteil von P_i mit der Darstellung (Präsentation) der Ein- und Ausgabedaten auf dem Bildschirm oder dem Drucker befaßt. In vielen Anwendungsprogrammen ist mehr oder weniger das gleiche „Bildschirmlayout“ für die Daten und die Interaktionensmöglichkeiten codiert. Es werden daher für diese Programmteile geeignete Hilfsmittel (Maskengeneratoren) bereitgestellt. Der Präsentations- und Interaktionsanteil von P_i wird ausgelagert und mittels solcher Generatoren einheitlich für das ganze Unternehmen realisiert. Die Anwendungsentwicklung konzentriert sich auf den verbleibenden Verarbeitungskern der Anwendung P'_i.

Mit der Möglichkeit neben alphanumerischen Zeichen auch farbige Graphiken auf dem Bildschirm darstellen zu können (zum Beispiel anklickbare Icons), ist die Präsentation der Daten zunehmend einem *graphical user interface tool* (GUI-Werkzeug) übertragbar. Die GUI--Werkzeugmöglichkeiten eine Anwendung zu gestalten wachsen über das reine Darstellen hinaus. Die Art und Weise wie zum Beispiel einzelne Funktionen oder Hilfen appliziert werden, wird mit dem GUI-Werkzeug von dem einzelnen P_i entkoppelt und unternehmensweit einheitlich geregelt.

GUI

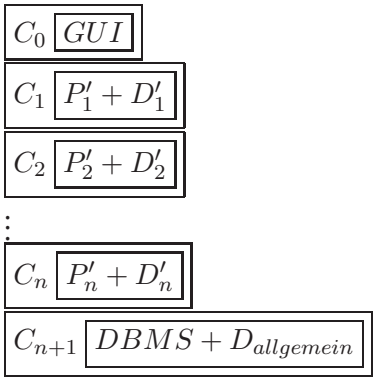
Damit ergeben sich drei Schichten, nämlich die Präsentationsschicht, die Applikationsschicht und die Datenbankschicht. In XML läßt sich somit notieren:

```
<!ELEMENT DREISCHICHTEN-SYSTEM
```

[22] Imperia CMS ↪ `http://www.imperia.de/` (online 24-Oct-2003) Imperia AG

```
  (C, PRÄSENTATION, APPLIKATION+, DATENBANK)>
<!ELEMENT PRÄSENTATION (GUI-TOOL, D)>
<!ELEMENT GUI-TOOL (P, D?)>
<!ELEMENT APPLIKATION (P, D?)>
<!ELEMENT DATENBANK (DBMS, D)>
<!ELEMENT DBMS (P, D?)>
```

1.2.5 Arbeitsteilung zwischen Computern: LAN-System



Der wachsende Bedarf an Computerleistung macht eine Verteilung auf mehrere Computer in einem Netzwerk notwendig. Zunächst können die Computer nicht weit von einander entfernt stehen, damit eine hinreichend schnelle Übertragung der Daten zwischen den Computern möglich ist. Man bezeichnet den Computerverbund daher als *Local Area Network* (LAN[23]).

LAN

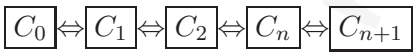
Mehrere relativ kleine Computer sind häufig, nicht immer kostengünstiger als ein großer Universalcomputer, kurz *Host* oder *Mainframe* genannt. Zunächst wird das $DBMS$ auf einen eigenen Computer, dem sogenannten Datenbank-*Server*, ausgelagert. Zunehmend werden auch die einzelnen Anwendungen auf sogenannte Applikations-*Server* verlagert und selbst die Aufbereitung der Daten wird „GUI-*Server(n)*“ übertragen. Merkformel gemäß formuliert: Jede Schicht $=$ ein eigener Computer.

```
<!ELEMENT LAN-SYSTEM (PRÄSENTATIONS-C+,
```

[23] Ein weitverbreitetes Protokoll, das die Datenübertragung in lokalen Netzwerken regelt, ist das sogenannte *Ethernet*. Der kalifornische Konzern *Xerox* hat diesen Übertragungsstandard 1976 entwickelt.

```
   APPLIKATIONS-C+, DATENBANK-C+, NETZ)>
<!ELEMENT PRÄSENTATION-C (C, GUI-TOOL)>
<!ELEMENT APPLIKATIONS-C (C, APPLIKATION)>
<!ELEMENT DATENBANK-C (C, DATENBANK)>
<!ELEMENT NETZ (KOMMUNIKATIONSKANAL+)>
```

1.2.6 Datenfernverarbeitung: WAN-System



Mit der Verbindung von verschiedenen Standorten zu einem größeren Verbund, genannt *Wide Area Network* (WAN), gewinnen die Kosten für die Datenfernübertragung (DFÜ) an Gewicht. Beispielhaft zeigt die Tabelle A.2 auf Seite 258 die Namen und *Internet Protocol* Adressen (IP) der betroffenen DFÜ-Computer bei einer Verbindung zum Web-Server `www.audi-usa.com`[24]. Bezogen auf die DFÜ-Kosten sind die Kosten für zusätzlichen Magnetspeicherplatz relativ gering. Die Aufgabe des $DBMS$ verschiebt sich. Die Vermeidung von Redundanz ist nicht mehr das Hauptziel. Es geht um die Gesamtkostenoptimierung bei örtlich weit verteilter Datenhaltung. Die verteilten Datenbankserver zusammen sollen diesen kostenoptimierten Zugriff gewährleisten, wobei Datenredundanz in den einzelnen Netzknoten die DFÜ-Kosten wesentlich senken kann. Ziel ist ein verteiltes (*distributed*) $DBMS$, das diese Optimierungsaufgabe löst. In der XML-Notation wird hervorgehoben, daß bei der `DATENFERNVERARBEITUNG` das Element `PROTOKOLL` bedeutsam ist.

DFÜ-Kosten

```
<!ELEMENT WAN-SYSTEM
  (DATENFERNVERARBEITUNG, LAN-SYSTEM+)>
<!ELEMENT DATENFERNVERARBEITUNG (PROTOKOLL, D)>
<!ELEMENT PROTOKOLL (P+)>
```

Die „leitungslose“ Datenferverarbeitung findet häufig zwischen Computern mit sehr unterschiedlichen Leistungspotentialen statt, zum Beispiel

Wireless

[24] Eine Internetadresse besteht aus einer Ziffernfolge. Damit man sie sich leichter merken kann, wird sie üblicherweise durch einen Namen (Zeichenfolge) ersetzt. Sogenannte *Domain Name Server* vollziehen die Übersetzung.

zwischen einem WAP-Gateway-Computer[25], einem (Origin[26]) Server und einem WAP-fähigem mobilen Telefon (Handy). Das WAP-Konzept basierend auf dem *Wireless Application Protocol* (WAP[27]) und der *Wireless Markup Language* (WML[28]). Es berücksichtigt die begrenzte Übertragungskapazität und die Speicher- und Darstellungsrestiktionen beim Endgerät (Handy). Das Bild 1.8 auf Seite 47 zeigt ein WAP-Beispiel.[29] Zukünftig werden mit dem Übergang zur UMTS-Technik[30] (Universal Mobile Telecomunication System) die Übertragungsrestriktionen wesentlich abgebaut.

1.2.7 Heterogene Infrastruktur: Plattformmix-System

$$C_j^1 \vee C_j^2 \vee \ldots \vee C_j^m \boxed{P_i' + D_i'}$$

Plattform

Netzwerke bestehen nur selten aus Computern vom gleichen Typ mit dem gleichen Betriebssystem. Vielmehr sind verschiedene Computerhersteller und verschiedene Betriebsysteme vertreten. Die Kombination bestehend aus Computer, Betriebssystem und wesentliche Softwareentwicklungswerkzeuge ($\equiv$ Tools) bezeichenet man als Plattform. Die international vertriebenen $DBMS$ und GUI-Werkzeuge sind auf unterschiedlichsten Plattformen verfügbar und können zunehmend plattformübergreifend „zusammenarbeiten“. Auch der Verarbeitungskern einer Anwendung P_i' muß auf unterschiedlichen Plattformen ablaufen können.

```
<!ELEMENT PLATTFORMMIX-SYSTEM (PLATTFORM+)>
<!ELEMENT PLATTFORM
```

[25] Ein WAP-Gateway umfaßt Software zum Aufbereiten (*Encoders* und *Decoders*) von Anfragen (*Requests*) und Anworten (*Responses*).

[26] Als *Origin Server* wird der Computer bezeichnet, auf dem die Daten sich ursprünglich befinden also sogenannt „residieren“.

[27] Näheres zum WAP-Konzept zum Beispiel:
↪`http://www.wapforum.org` Zugriff 05-Jun-2000
und ↪[Hei00]

[28] WML ist eine Seitenbeschreibungssprache, die speziel für den Einsatz auf mobilen Geräten entwickelt wurde. Sie entspricht dem, was HTML für die herkömmlichen Webseiten ist. Näheres zu WML zum Beispiel:
↪`http://www.wapforum.org/DTD/wml_1.1.xml` Zugriff 03-Jun-2000

[29] Hier kommt es nicht auf das Verstehen der WML-Konstrukte an. Das Bild soll nur die Darstellungsrestriktionen veranschaulichen.

[30] Näheres zu UMTS zum Beispiel ↪[Hol+00].

```
 1  <?xml version="1.0" encoding="ISO-8859-1"?>
 2  <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//E
 3    "http://www.wapforum.org/DTD/wml_1.1.xml">
 4  <wml>
 5  <!-- Bonin 28-Jun-2000 ... 29-Sep-2000  -->
 6    <template>
 7      <do type="accept" label="Impressum" name="imp
 8        <go href="#impress"/>
 9      </do>
10    </template>
11    <card id="start" title="FHNON"
12      ontimer="#menu">
13      <timer value="100"/>
14      <do type="accept" label="Impressum" name="imp
15        <noop/>
16      </do>
17      <p align="center">(Denk-)Welt</p>
18      <p align="center">der</p>
19      <p align="center">Wirtschafts-</p>
20      <p align="center">Informatik</p>
21    </card>
22    <card id="menu" title="WI-Elemente">
23      <p><a href="#hardware">Hardware</a></p>
24      <p><a href="#software">Software</a></p>
25      <p><a href="#org">Organisation</a></p>
26      <p><a href="#user">Nutzer+Betroffene</a></p>
27    </card>
28    <card id="hardware" title="Hardware">
29      <do type="prev" label="zurueck">
30        <prev/>
31      </do>
32      <p>Das EVA-Modell:<br/>
33         Eingabe-Verabeitung-Ausgabe</p>
34    </card>
35    <card id="software" title="Software">
36      <do type="prev" label="zurueck">
37        <prev/>
38      </do>
39      <p>Programme+Dokumentation</p>
40    </card>
41    <card id="org" title="Organisation">
42      <do type="prev" label="zurueck">
43        <prev/>
44      </do>
45      <p>Flexibilitaet+Vereisungseffekt</p>
46    </card>
47    <card id="user" title="Nutzer+Betroffene">
48      <do type="prev" label="zurueck">
49        <prev/>
50      </do>
51      <p>Interessengepraegte Rollen</p>
52    </card>
53    <card id="impress" title="Impressum">
54      <do type="accept" label="Impressum" name="imp
55        <noop/>
56      </do>
57      <do type="prev" label="zurueck">
58        <prev/>
59      </do>
60      <p>
61        Hinrich E.G. Bonin<br/>
62        <small>Department of Business Administratio
63        University of Applied Sciences NON Germany<
64        http://as.fh-lueneburg.de<br/>
65        28-Jun-2000</small>
66      </p>
67    </card>
68  </wml>
69
```

Legende:

WAP ≡ Wireless Application Protocol

WML ≡ Wireless Markup Language

Bild erzeugt mit *YOURWAP* Release 1.16, Wireless Companion

Abbildung 1.8: WAP-Beispiel: Darstellung eines Ausschnittes einer WML-Datei auf einem Handy

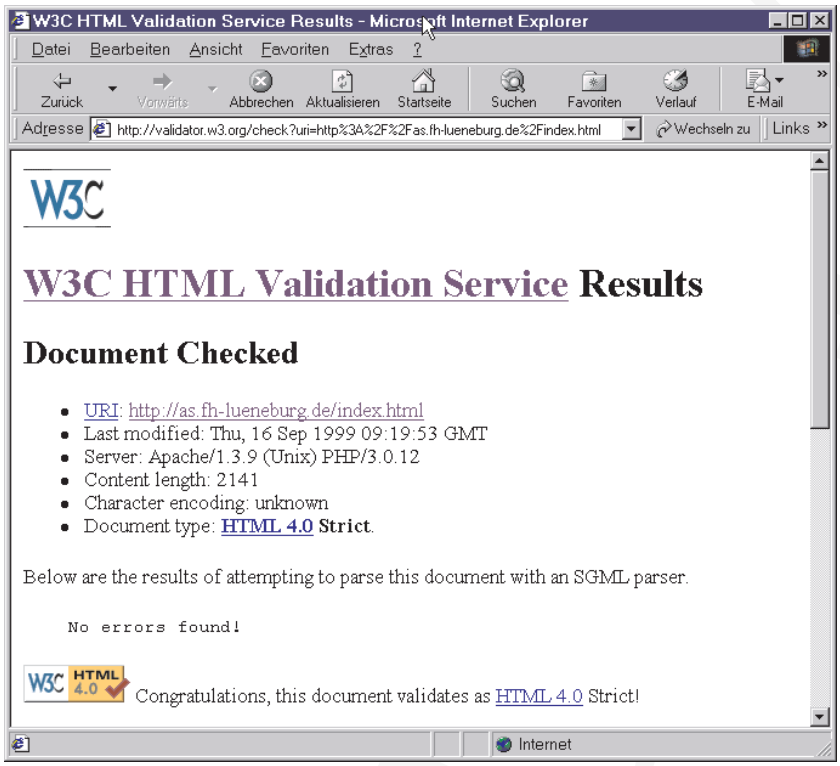


Abbildung 1.9: Validierung eines Web-Dokuments in Bezug auf die Einhaltung des Standards HTML 4.0

```
  (C, BETRIEBSSYSTEM, TOOL+)>
<!ELEMENT BETRIEBSSYTEM (P, D)>
<!ELEMENT TOOL (P, D)>
```

Normen und Standards, die von möglichst vielen Knoten im Netz erfüllt werden, prägen P_i'. Wichtiges Leistungskriterium für P_i' wird die Plattformunabhängigkeit. Die norm- und standardeinhaltende Programmierung wird zunehmend bedeutsamer als die Realisierung von Effizienz oder das Ausschöpfen des maximal möglichen Leistungsspektrums der Anwendungssoftware. Dabei muß die Einhaltung der Normen und Standards konkret geprüft werden; möglichst durch unabhängige Dritte. Ein Beispiel einer Validierung zeigt Bild 1.9 auf Seite 48.

Validierung

1.2.8 Prägender Trend: *Nomadic Computing*

Diese grobe Skizze der WI-Entwicklung wäre unvollständig ohne eine Prognose der zukünftigen Technik. Eine solche Vorhersage ist jedoch sehr risikoreich — wie die Vergangenheit gezeigt hat: Weder haben heute alle Haushalte einen Computer, der den Tisch abräumt, wie in den 80iger Jahren geträumt wurde, noch sind alle Programme in der Sprache PROLOG[31] (*Programming in Logic*) geschrieben, der regelbasierten Programmiersprache, die schon die fünfte Generation von Computern beherrschen sollte. Trotz alldem sei hier prognostiziert, daß der prägende WI-Trend das sogenannte *Nomadic Computing*[32] ist, also ein Überall- und Jederzeit-Computing.[33]

Überall

Jederzeit

Das bedingt einen Ausbau von leitungslosen Netzwerken für mobile und „standorttreue“ Computer. Nicht nur die Vielfalt der mobilen Endgeräte, zum Beispiel Handy oder *Personal Digital Assistant* (PDA), sondern auch die Server und speziellen Endgeräte in einem Bürogebäudekomplex werden leitungslos verbunden um ihren Aufstellungsort leicht verändern zu können. Dazu werden diese leitungslosen Netzwerke für den einzelnen Teilnehmer auch eine große Übertragungskapazität (Bandbreite) bereitstellen. Die Teilnehmer werden durch ihre Software selbst feststellen, welches von den erreichbaren Netzwerken gerade die gewünschte Bandbreite, Qualität und die geringsten Kosten bietet. Beispielsweise wird ein PDA verbindbar mit mehreren LANs (z. B. als Intranet & Internet) und WANs (z. B. mittels UMTS[34]). Dabei werden die verschiedenen leitungslose Netzwerke zukünftig mit einer einheitliche Identifizierung (Netzadresse) arbeiten.

Der „personenbezogene Computer“ der Zukunft wird ein winziger, aber mächtiger Computer sein, der zusätzlich die Funktionen des Telefonierens mit jenen eines Fotoapparates und einer Videokamera verknüpft, der über ein präzises, generelles Positionierungssystem mit zusätzlichen

[31] Einen Überblick über Programmiersprachen ↪Tabelle 1.1 Seite 62.

[32] Der Begriff *Nomadic Computing* wurde von L. Kleinrock gewählt — Nomadic computing: An opportunity, in: Computer Communications Review, Jan. 1995; zitiert nach [Var+00].

[33] Vielleicht tragen dazu auch die preiswerten Chips auf RFID-Technologie (Radiofrequenzidentifikation) bei, die eine Identifikation, Ortsverfolgung und Zustandsüberwachung von Alltagsdingen kostengünstig ermöglichen.

[34] Näheres zu UMTS zum Beispiel:
↪`http://www.umts-forum.org` Zugriff 15-Jul-2000

Sensoren verfügt, die zum Beispiel auch die Position des Kopfes des Benutzers ermitteln, inclusive Blickrichtung und Kopfneigung (↪ [Mau04], S. 44). Ein solcher Computer wird in Zukunft (— im Jahre 2014 — ?[35]) als elektronische Identifikation im Sinne eines Personalausweises, Reisepasses und Führerscheins eingesetzt werden. Er wird anstelle von Kreditkarten zum Zahlen, als Schlüssel für Türen oder Safes ausgelegt sein und wird viele andere Steuerungsfunktionen übernehmen. Er wird keine Harddisk haben, keinen Bildschirm, keine Tastatur und keinen Akku, wie wir sie im Jahr 2004 kennen. Ob faltbare, knitterbare Schirme, die in die Kleidung eingebaut sein könnten, oder Projektoren, die auf jede (auch unebene und farbige) Flächen einwandfreie Bilder projizieren, oder Brillen mit Spiegelchen, direkt durch die Pupille auf die Retina der Augen projizieren, sich durchsetzen, ist kaum vorhersehbar. Die Tastatur wird primär durch Spracheingabe ersetzt werden. Hinzukommen unorthodoxe Eingabegeräte, die beispielsweise Kopfbewegungen als Eingabesignale verwenden oder über die Videokamera Gesten interpertieren (↪ [Mau04], S. 44).

IT-Sicherheit

Im Kontext von *Nomadic Computing* mit seiner Ubiquität von Daten kommt der Frage der IT-Sicherheit eine besondere Bedeutung zu. Einerseits gilt es den klassischen Datenschutz, insbesondere von personenbezogenen Daten, zu gewährleisten. Andererseits muß dem Zuschütten mit unerwünschten Daten („Müll“, *Spam*[36]) begegnet werden. Dabei könnten auch neue Verhaltensweisen der Benutzer eine Schutzfunktion übernehmen. Ein Ansatz dazu bildet das *Wiki*-Konzept.[37]

[35] Hermann Maurer bezeichnet diesen Computer daher als PC14 ↪ [Mau04].

[36] *Spam* ≡ elektronisches Äquivalent unerwünschter Wurfsendungen

[37] Der Begriff *Wiki* ist vom hawaiischen *wiki wiki* abgeleitet und bedeutet frei übersetzt ungefähr *schnell*. Auf sogenannten *Wiki*-Web-Seiten kann jeder Besucher anonym Änderungen vornehmen und eigene Beiträge hinzufügen. Der Programmierer Ward Cunningham gilt als Begründer des *Wiki*-Konzeptes. 1995 stellte er mit dem *Portland Pattern Repository* die erste *Wiki*-Seite ins Web und schuf damit eine offene Wissensdatenbank. Inzwischen gibt es eine Vielzahl von frei verfügbarer *Wiki*-Software; siehe beispielsweise `http://www.twiki.org/` oder `http://c2.com/cgi-bin/wiki?WikiEngines` (online 23-May-2003) und zwar auch in Programmiersprachen, die nicht so „in Mode sind“ wie Java^{TM}. Ein solches Beispiel ist die Software *MoshiMoshi*, geschrieben in dem LISP-Abkömmling `Scheme` ↪`http://schematics.sourceforge.net/moshimoshi.html` (online 23-May-2003).

1.3 WI-Elemente

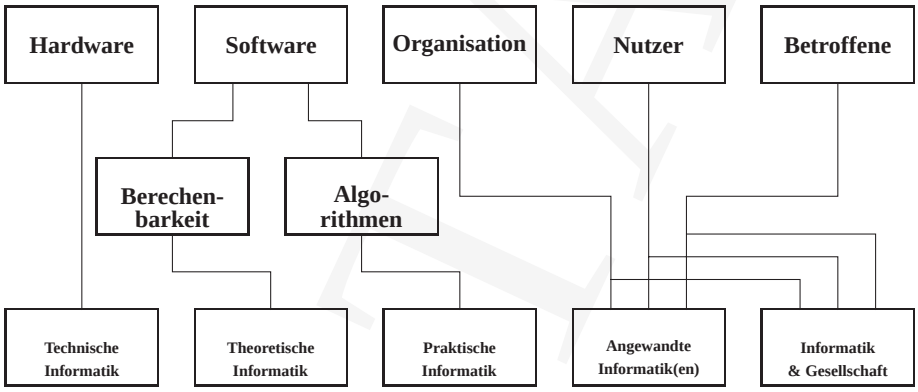
Die WI ist unstrittig eine interdisziplinäre Disziplin. Der beteiligte Fächerkanon ist kaum begrenzt. Dazu zählen beispielsweise auch die Psychologie und die Rechtswissenschaften.[38] In der Datei `wikanon.xml` sind beispielhaft Beiträge der (zumindest) beteiligten Disziplinen stichwortartig genannt. Die drei Punkte deuten an, daß es sich nicht um eine vollständige Aufzählung handeln kann.

`wikanon.xml`

```
 1  <?xml version="1.0" encoding="ISO-8859-1"?>
 2  <!-- Bonin    14-Dec-1999              -->
 3  <!DOCTYPE WI [
 4    <!ELEMENT WI (BWL, INFORMATIK, ELEKTROTECHNIK,
 5      MATHEMATIK, STATISTIK, OPERATIONSRESEARCH,
 6      POLITIK, RECHT, PSYCHOLOGIE)>
 7    <!ELEMENT BWL (#PCDATA)>
 8    <!ELEMENT INFORMATIK (#PCDATA)>
 9    <!ELEMENT ELEKTROTECHNIK (#PCDATA)>
10    <!ELEMENT MATHEMATIK (#PCDATA)>
11    <!ELEMENT STATISTIK (#PCDATA)>
12    <!ELEMENT OPERATIONSRESEARCH (#PCDATA)>
13    <!ELEMENT POLITIK (#PCDATA)>
14    <!ELEMENT RECHT (#PCDATA)>
15    <!ELEMENT PSYCHOLOGIE (#PCDATA)>
16  ]>
17  <WI>
18    <BWL>
19      Gestaltung betrieblicher Funktionsbereiche
20      Entscheidungslehre
21      Unternehmensübergreifende Organisationsstrukturen
22      ...
23    </BWL>
24    <INFORMATIK>
25      Softwaretechnik
26      Datenbankmanagementsysteme
27      Artificial Intelligence
28      Visualisierungstechnik
```

[38] Beide Disziplinen integriert zum Beispiel Peter Mertens, einer der „WI-Gründungsväter“ (↪[Me+04]).

```
29      ...
30    </INFORMATIK>
31    <ELEKTROTECHNIK>
32      Hardwarebausteine
33      Kommunitkationstechnik
34      ...
35    </ELEKTROTECHNIK>
36    <MATHEMATIK>
37      Datenkomprimierung
38      Datenverschlüsselung
39      Algorithmenoptimierung
40      ...
41    </MATHEMATIK>
42    <STATISTIK>
43      Marktforschung
44      Qualitätssicherung
45      Data Mining
46      ...
47    </STATISTIK>
48    <OPERATIONSRESEARCH>
49      Logistikoptimierung
50      Verbrauchsminimierung
51      Kapazitätsplanung
52      ...
53    </OPERATIONSRESEARCH>
54    <POLITIK>
55      Innovationsförderung
56      ...
57    </POLITIK>
58    <RECHT>
59      Zweckentsprechende Datennutzung
60      Informationelle Gewährleistungsarchitektur
61      Copyrights
62      Vertragsgestaltung
63      Produkthaftung
64      ...
65    </RECHT>
66    <PSYCHOLOGIE>
67      Systemakzeptanz
68      Handhabbarkeit von Systemen
69      ...
70    </PSYCHOLOGIE>
```



Legende:
$\boxed{y}$—$\boxed{x}$ ≡ Assoziation; hier im Sinne x befaßt sich primär mit y

Abbildung 1.10: Spezialinformatik(en) für die WI-Elemente — in UML

```
71  </WI>
72  <!-- End of object wikanon.xml          -->
73
```

WI>Data konzentriert sich auf die Elemente: Hardware, Software, Organisation, Nutzer und Betroffene. Dabei ist klar, daß es für diese Elemente jeweils auch Spezialdisziplinen der (oder in der) Informatik gibt. Das Bild 1.10 auf der Seite 53 ordnetet den Elementen den Spezialdisziplinen zu.

1.3.1 Hardware: Das „EVA“-Modell

Der Begriff Hardware bezeichnet die Menge aller technischen Geräte eines Computers. Zur Hardware gehören daher der Computerkern (*Central Processing Unit*[39] (CPU)), die internen und externen Speicher (Arbeitsspeicher und Magnetplattenspeicher) und die Eingabe- und Ausgabegeräte wie zum Beispiel Bildschirm, Drucker, Scanner, Videocamera, Mikrophon und Netzanschlüsse. Die Weiterentwicklung ist gerade im Bereich der Hardware besonders schnell. Laufend verbessert sich die Möglichkeit große Datenmengen kostengünstig zu speichern. Stellvertretend für viele Entwicklungen bei den Datenspeichern seien hier die

CPU

[39] Zentraleinheit ≡ Computerkern, der „Alles“ umfaßt, um Befehle interpretieren und ausführen zu können.

PC-Karten nach dem PCMCIA[40]-Standard und die *Digital Versatile Disc* (DVD[41]) genannt. Eine PCMCIA-Karte vom Typ 3 bietet eine Speicherkapazität bis in den GigaByte-Bereich. Die vielseitige DVD-Scheibe hat im Vergleich zur *Audio Compact Disc* (Audio-CD) oder Daten-CD (*Read Only Memory CD*; kurz: CD-ROM) eine bis zu 25-mal hähere Speicherkapazität. Eine DVD kann bis zu 8 Stunden Film in brillanter Bild- und Tonqualität aufnehmen.

Die klassische Gliederung im Bereich Hardware fußt auf dem„EVA-Modell[42]“ (↪Bild 1.11 auf Seite 56). Es unterteilt den Ablauf in die drei Phasen:

Peripherie

1. Phase: Eingabe von Daten
2. Phase: Verarbeitung von Daten
3. Phase: Ausgabe von Daten

Die Begriffe „Eingabe“ (Input) und „Ausgabe“ (Output) bezeichnen die Flußrichtung von Daten. Dabei geht es allgemeiner formuliert um eine „Hole-Daten“-Operation (≡ Get-Operation) und um eine „Sende-Daten“-Operation (≡ Put-Operation). Der Begriff „Verarbeitung“ (Process) von Daten ist weitergehend als die Modifikation eines Speicherinhaltes. Er umfaßt das Lesen eines Speicherplatzes und das Transportieren der Daten von einem Speicherplatz zu einem anderen. Detaillierter betrachtet entspricht die „Verarbeitung“ einer Menge von Ein- und Ausgabe-Operationen von einzelnen Bits im internen Bereich. So betrachtet läßt sich Hardware in XML wie folgt beschreiben:

```
<!ELEMENT HARDWARE (INPUT, PROCESS, OUTPUT)>
<!ELEMENT INPUT ANY>
<!ELEMENT PROCESS (MEMORY, PROCESSORUNIT)>
<!ELEMENT MEMORY ANY>
```

[40] PCMCIA ≡ *Personal Computer Memory Card International Association*. Eine PCMCIA-Karte hat die Größe einer Kreditkarte ($36mm \star 43mm$) und einen 68-poliges Interface. Man unterscheidet 3 Typen, die jeweils unterschiedliche Dicke haben (Typ $1 = 3,3mm$; Typ $2 = 5mm$; Typ $3 = 10,5mm$). Näheres dazu ↪`http://www.pcmcia.org/about.htm` Zugriff 16-Dec-1999.

[41] Näheres zur DVD-Technik zum Beispiel ↪`http://www.dvdforum.org/techa.htm` Zugriff 10-Dec-1999.

[42] engl.: IPO ≡ *input, process, output*

```
<!ELEMENT PROCESSORUNIT ANY>
<!ELEMENT OUTPUT ANY>
```

Dabei definiert die Angabe ANY die Möglichkeit von beliebigen Datenformaten; also auch andere als #PCDATA-Daten[43]. EVA

1.3.2 Software: Programme & Dokumentation

Der Begriff *Software* gehört heute zum Alltagssprachgebrauch. Er ist daher unscharf und wird mit unterschiedlichen Bedeutungen benutzt. Allgemein bezeichnet man mit Software eine Menge von Programmen mit und ohne Daten zusammen mit den Dokumenten, die für die Anwendung notwendig und nützlich sind. Software ist damit Programm plus Dokumentation.

Software im engeren Sinne sind alle Texte, die den potentiell universellen Computer für eine bestimmte Aufgabe spezifizieren, insbesondere diejenigen Texte, die als Instruktionen (Anweisungen, Befehle, Kommandos, Statements etc.) der Veränderung von Zuständen (Daten) dienen; also kurz: Programme. Dabei läßt sich ein Programm abstrakt betrachten als eine statische Spezifikation dynamischer Abläufe. Software

Unter *Software im weiten Sinne* werden Programme einschließlich ihrer Programmdokumentation plus alle vorbereitenden Dokumente (Problemuntersuchung, Anforderungsanalyse, Entwürfe, . . .) plus alle nachbereitenden Dokumente (Testprotokolle, Integrationspläne, Installationsanweisungen, . . .) verstanden. Diese weite Auslegung fußt nicht nur auf dem Produkt „Software“, sondern bezieht auch den Prozeß der Entwicklung und der Fortschreibung (Erweiterung, Wartung & Pflege) des Produktes – zumindest zum Teil – ein.

```
<!ELEMENT SOFTWARE (PROGRAMM+, DOKUMENTATION+)>
<!ELEMENT PROGRAMM (ABLAUFSPEZIFIKATION)>
<!ELEMENT ABLAUFSPEZIFIKATION (#PCDATA)>
<!ELEMENT DOKUMENTATION
  (PRODUKTDOKUMENT+, PROZESSDOKUMENT+)>
<!ELEMENT PRODUKTDOKUMENT (PROGRAMMBESCHREIBUNG+)>
<!ELEMENT PROZESSDOKUMENT
  (VORBEREITUNGSDOKUMENT+,
```

[43] Beschreibung von #PCDATA ↪Seite 28

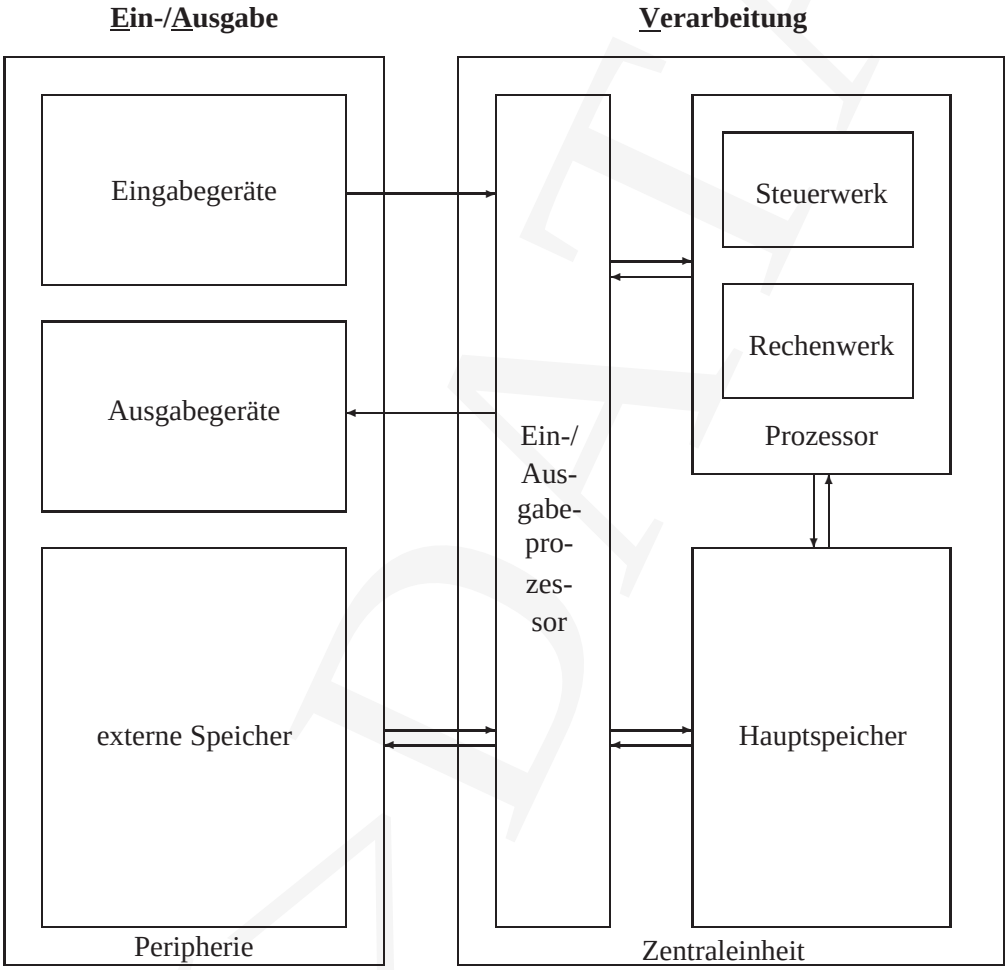


Abbildung 1.11: EVA-Modell: Eingabe↩Verarbeitung↩Ausgabe

```
  NACHBEREITUNGSDOKUMENT+)>
<!ELEMENT VORBEREITUNGSDOKUMENT
  (PROBLEMANALYSE+, ANFORDERUNGSANALYSE+,
  GROBENTWURF+, FEINENTWURF+)>
<!ELEMENT NACHBEREITUNGSDOKUMENT
  (TESTPROTOKOLL+, INTEGRATIONSPLAN+,
  INSTALLATIONSANWEISUNG+)>
```

Programm

Auf der Grenzlinie zwischen Software und Hardware liegt die Firmware. Sie bezeichnet alle in einem Prozessor realisierten Mikroprogramme, die den Befehlsvorrat des Prozessors nach außen realisieren. Diese Mikroprogramme sind zwar wie jede Software prinzipiell leicht änderbar, bleiben jedoch in der Praxis für den Lebenszyklus eines bestimmten Prozessortyps unverändert[44] und werden daher auch der Hardware „zugerechnet“.

Algorithmus

Im Mittelpunkt der Software steht der Begriff „Algorithmus“. Darunter versteht man ein eindeutig bestimmtes Verfahren (Prozeß) oder die Beschreibung eines solchen Verfahrens. Diese intuitive Definition[45] verdeutlicht, daß es bei einem Algorithmus nicht nur um ein Verfahren geht, sondern auch um seine Beschreibung. Sie kann in einer natürlichen oder formalen Sprache (zum Beispiel in Java^{TM}) oder mit Hilfe eines graphischen Darstellungsmittel (zum Beispiel in UML) erfolgen. Bedeutsam ist die Eindeutigkeit der Beschreibung mit Hilfe eines endlichen Textes. Bestimmt wird ein Algorithmus durch die Vorgabe:

Firmware

- einer Menge von Eingabe-, Zwischen- und Endgrößen,
- einer Menge von Elementaroperationen und
- der Vorschrift, in welcher Reihenfolge welche Operationen wann auszuführen sind.

[44] Änderungen führt eigentlich nur der Hardwarehersteller durch.

[45] Bei den theoretisch fundierten Definitionen des Algorithmusbegriffs zum Beispiel von A. M. Turing oder A. A. Markow geht es um Aufzählbarkeit, Berechenbarkeit und Endscheidbarkeit.

In der Softwareentwicklungspraxis, zum Beispiel beim Abbilden einer Kostenrechnung, spricht man eher vom „Programm“ als vom „Algorithmus“. Dabei unterscheidet man strikt zwischen den Daten (≡ Eingabe-, Zwischen- und Endgrößen) und dem Programm (≡ Vorschrift basierend auf Elementaroperationen), obwohl im ursprünglichen Computer, der sogenannten „von-Neumann-Maschine“[46], die Trennungslinie zwischen Programm und Daten unscharf ist. Man verbindet mit dem Begriff „Programm“ etwas Aktives wie Datenerzeugung, Datenmodifikation oder Datenlöschung. Die Daten selbst sind dabei passiv. Sie werden vom Programm manipuliert.

In groben Strichen skizziert sieht ein Algorithmus folgendermaßen aus:

```
<!ELEMENT ALGORITHMUS
  (ZUSTAND+, ELEMENTAROPERATION+, VORSCHRIFT)>
<!ELEMENT ZUSTAND
  (EINGABEGROESSE+, ZWISCHENGROESSE+, ENDGROESSE+)>
```

Softwarearten

Eine Klassifizierung von Software kann von den unterschiedlichsten Gesichtspunkten ausgehen. Eine übliche Einteilung in der WI unterscheidet die hardwarenahe Software (≡ Systemsoftware) von der anwendernahen Software (≡ Anwendungssoftware).[47] Der ersten Art zugeordnet werden Betriebssysteme, Übersetzungsprogramme (*Compiler*) und „Dienstprogramme“, wie zum Beispiel *File Transfer Programme* (FTP) oder Sortierprogramme. Die Anwendungssoftware wird nach dem Grad der Standardisierung beziehungsweise nach der Individualität weiter unterschieden. Formuliert in XML ergibt sich dann folgende pragmatische Einteilung:

endlicher Text

```
<!ELEMENT SOFTWARE
  (SYSTEMSOFTWARE, ANWENDUNGSSOFTWARE)>
<!ELEMENT SYSTEMSOFTWARE
  (BETRIEBSSYSTEM, WERKZEUG)>
```

[46] John von Neumann hat im Jahre 1946 ein Konzept zur Gestaltung eines universellen Rechners vorgeschlagen. Dabei werden Programme, Daten, Zwischen- und Ergebnisse in demselben Speicher abgelegt.

[47] Vergleiche zum Beispiel ↪[Me+04].

```
<!ELEMENT ANWENDUNGSSOFTWARE
  (STANDARDSOFTWARE, INDIVIDUALSOFTWARE)>
```

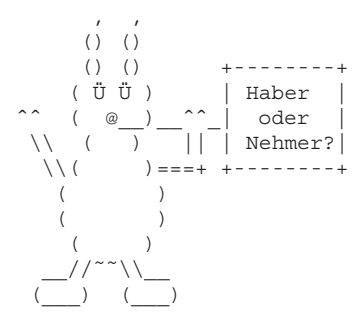
Bezieht sich eine Standardsoftware nicht auf eine bestimmte betriebliche Funktion, sondern ist übergreifend einsetzbar wie beispielsweise ein Textverarbeitungsprogramm, dann wird ebenfalls der Begriff Werkzeug (*Tool*) verwendet. Die Metapher „Werkzeug“ betont Einsatzalternativen und Nutzungsvarianten. Der einzelne Nutzer kann individuell mit dieser Software seinen Aufgabendurchführung gestalten.

Ein weitere WI-relevante Softwareklassifikation orientiert sich an der Betriebsform. Dabei wird unterschieden zwischen Stapelverarbeitung (*Batch Processing*) und interaktiver Verarbeitung (*Conversational Mode*).

Hardware-sicht

Eine Stapelverarbeitung ist gegeben, wenn der Auftrag des Nutzers vorab vollständig spezifiziert ist und als Ganzes erteilt wird bevor mit seiner Abwicklung begonnen wird (↪DIN 44300, Teil 9). Beispielsweise wird die Bilanz eines Unternehmens aus allen Geschäftsvorfällen des Jahres im Stapelbetrieb berechnet. Bei der interaktiven Verarbeitung kommt es zu einem ständigen Wechsel von Nutzereingaben und Systemausgaben. Die Software wickelt eine Kette von Teilaufträge ab, die jeweils situationsabhängig erteilt werden. Bei der interaktiven Verarbeitung unterscheidet man die Echzeitverarbeitung (*Real Time Processing*) von der Dialogverarbeitung. Bei der Echzeitverarbeitung muß das Programm in einer vorher geplanten Maximalzeit auf ein Ereignis (zum Beispiel Signal eines Sensors) garantiert reagieren und zwar unabhängig von der aktuellen Arbeitslastung des Computers. In Echzeitverarbeitung werden Prozesse in technischen Anlagen wie zum Beispiel in Kraftfahrzeugen, Waschmaschinen oder Flugzeugen gesteuert. Man spricht daher auch von Prozeßverarbeitung.

Stapel



Bei der Dialogverarbeitung unterscheidet man, ob mit dem Computer ein Nutzer oder mehrere (quasi) gleichzeitig arbeiten. Im *Multiuser*-Betrieb kann das Programm sich für den einzelnen Betnutzer im Teilhaber- oder Teilnehmerbetrieb befinden. Beim Teilhaberbetrieb arbeiten mehrere Nutzer mit demselben Programm und denselben Basisdaten. Im Arbeitsspeicher des Computers ist dann nur ein Programm geladen, ein großer Bereich für die gemeinsamen Basisdaten reserviert und ein kleiner Bereich für die Speicherung der benutzerspezifischen Daten vorgesehen.

Real Time

Der Teihaberbetrieb ist üblich bei Buchungssystemen, die im *Online Transaction Processing* (OLTP) ihre Daten fortschreiben. Beispielsweise umfaßt eine Transaktion dann die konsitente Änderung der beiden Konten (Doppik).

Im Teilnehmerbetrieb ist für jeden einzelnen Nutzer im Arbeitsspeicher eine Kopie des Programms geladen und jeweils ein Datenbereich für den Nutzer reserviert. Formuliert in XML ergibt sich dann folgende pragmatische Einteilung:

OLTP

```
<!ELEMENT SOFTWAREBETRIEBSFORM
  (STAPEL | INTERAKTIV)>
<!ELEMENT INTERAKTIV (DIALOG | REALTIME)>
<!ELEMENT DIALOG (SINGLEUSER | MULTIUSER)>
<!ELEMENT MULTIUSER
  (TEILHABERBETRIEB | TEILNEHMERBETRIEB)>
<!ELEMENT TEILHABERBETRIEB
  (PROGRAMM, BASISDATEN, NUTZERDATEN+)>
<!ELEMENT TEILNEHMERBETRIEB
  (PROGRAMM, BASISDATEN, NUTZERDATEN)+>
```

Programmiersprache

Betriebsform

Praktisch gibt es eine unbegrenzte Menge von Programmiersprachen. Die Tabelle 1.1 auf der Seite 62 nennt einige, die für die WI bedeutsam waren oder sind. Um im Babylon der Programmiersprachen einen

Überblick zu behalten, kann man sich an der Problem(lösungs)sicht orientieren, für die eine Sprache besonders geeignet ist. Holzschnittartig lassen sich dabei folgende Sichtweisen herauskristallisieren:

- imperative Sicht ≡ Die Aufgabe wird mit Befehlen zur Zustandsänderung gemeistert.
  Sprachbeispiel: COBOL (*Common Business-oriented Language*)
  Quellcodebeispiel:
  ```
  konto = konto + betrag
  anzeigeFeld = konto
  ```

- funktionale Sicht ≡ Die Aufgabe wird mit Funktionen, ähnlich einer mathematischen Funktionsverknüpfung $g(f(x))$, gemeistert.
  Sprachbeispiel: LISP (*List Processing*)
  Quellcodebeispiel:
  ```
  anzeigen(buchen(konto,betrag))
  ```

COBOL

- objekt-orientierte Sicht ≡ Die Aufgabe wird mit kommunizierenden Objekten gemeistert.
  Sprachbeispiel: Java
  Quellcodebeispiel:
  ```
  konto.buchen(betrag);
  konto.anzeigen();
  ```

LISP

Da (beinahe) alle Programmiersprachen die imperative Sicht ermöglichen, wird die imperative Ausrichtung als Default-Wert in der folgenden XML-Notation angegeben:

```
<!ELEMENT PROGRAMMIERSPRACHE ANY)>
  <!ATTLIST PROGRAMMIERSPRACHE AUSRICHTUNG
    (IMPERATIV | FUNKTIONAL | OBJEKTORIENTIERT)
    "IMPERATIV">
```

Eine weitere WI-relevante Einteilung von Programmiersprachen bezieht sich auf die Größe (Komplexität) des Programms. Man unterscheidet Sprachen für das Programmieren von kleinen Systemen (*Programming-in-the-Small*) von denjenigen für das Programmieren von großen Systemen (*Programming-in-the-Large*). Zur ersten Kategorie zählt beispielsweise BASIC (*Beginners All Purpose Symbolic Instruction Code*), zur zweiten COBOL (*Common Business-oriented Language*).

Ein Programm ist in einer formal(isiert)en Sprache geschrieben. Wie jede Sprache, so dient auch eine Programmiersprache der Kommunikation. Ihre Aufgabe ist die Kommunikation zwischen Menschen über Texte

Sprache	Beginn	Erläuterung
FORTRAN	≈ 1955	Die Sprache *Formular Translation* ist eine der ersten Programmiersprachen mit einer Ausrichtung auf mathematisch-naturwissenschaftliche Probleme.
LISP	≈ 1955	*List Processing Language* ist eine funktionale Sprache mit der Rekursion als prägendes Konzept. Im AI-Bereich (*Artificial Intelligence*) ist sie der „Klassiker". Auch wurde frühzeitig in CLOS (*Common Lisp Object System*) multiple Vererbung und Meta-Objekt-Programmierung realisiert.
COBOL	≈ 1960	*Common Business-oriented Language* ist in der aktuellen Fassung weiterhin ein „Arbeitspferd" für betriebswirtschaftliche Aufgaben.
ALGOL	≈ 1960	*Algorithmic Language* umfaßte als erste Sprache die grundlegenden Konzepte für die Programmierstile: imperativ, funktional, muster-, logik- und objekt-orientiert.
BASIC	≈ 1965	*Beginners All Purpose Symbolic Instruction Code* wird durch einen Interpreter ausgeführt. BASIC ermöglicht besonders auf Personalcomputern das schnelle Entwicklen von kleinen Programmen („Prototyping").
PL/I	≈ 1965	*Programming Language I* vereinige die Konzepte von FORTRAN und COBOL. Die Mächtigkeit von PL/I wurde zum Problem der Lernbarkeit und Durchschaubarkeit — daher kaum noch PL/I-Projekte.
Pascal	≈ 1970	Eine modere, verschlankte Sprache auf der Basis von ALGOL, die primär auf Personal Computern eingesetzt wird. Nachfolger sind Modula-2, Delphi und Oberon.
C/C++	≈ 1973	Die Sprache für systemnahe Programmierung im Umfeld von UNIX mit C++ als objektorientierte Variante.
Ada	≈ 1980	Ada, genannt nach Augusta Ada Byron, Countess of Lovelace (1815–1852), der Standardisierungsversuch der US-Militärs, ist eine Sprache mit Validierungskonzept, die auch im Bereich der Prozeßdatenverarbeitung einsetzbar ist.
SmallTalk	≈ 1980	Eine Sprache mit konsequenter Ausrichtung auf das objektorientierte Paradigma. Selbst die Addition $1 + 1$ wird als Nachrichtenaustausch aufgefaßt.
Java	≈ 1995	Java^{TM}, die objekt-orientierte Sprache des amerikanischen Unternehmens *Sun*, ist prädestiniert für Anwendungen im Netz.

Legende: Näheres dazu zum Beispiel ↪[Bo91b, Han96, Rech91, St+99]

Tabelle 1.1: Relevante Programmiersprachen für die WI-Entwicklung

(„Botschaften“), die ein Computer als Arbeitsanweisungen interpretieren und ausführen kann. Formulierungen in einer Programmiersprache haben einerseits Menschen, andererseits Computer als Adressaten. Jeder Text in einer Programmiersprache hat zumindest im Erstellungsprozeß (Zyklus: Entwurf → Formulierung → Test → Entwurf . . .) seinen Konstrukteur als Kommunikationspartner. Darüber hinaus wendet sich der Text an Dritte; an Mitkonstrukteure und an das Wartungs-/Pflege- und Fortentwicklungs-Personal. Programmieren ist damit das Erstellen von Texten aus (sprachlichen) Konstrukten, welche die Arbeit des Computers bestimmen.

Eine Programmiersprache kann stärker die Kommunikation mit dem Computer oder die zwischen Programmieren unterstützen. Ein in Maschinencode notiertes Programm kann das gesamte Leistungsvermögen („Verstehbarkeitspotential“) des Computers effizient ausnutzen. Ein Programmierer kann ein solches Programm in der Regel nur mühsam verstehen. Ein Text, in einer Assembler-Sprache formuliert, ist im allgemeinen aufwendiger zu durchschauen als ein äquivalenter Text, notiert in einer höheren Programmiersprache. Dafür wird der Assembler-Text als eine $1:1$-Abbildung des Maschinencodes vom Computer mit weniger Aufwand in konkrete Arbeitsschritte umgesetzt. Ein Programmiersprache ist daher stets ein Kompromiß zwischen den unterschiedlichen Anforderungen der Kommunikations-„Partner“ Mensch und Maschine.

Java

Transparenz des Programms

In einer natürlichen Sprache kann ein Sachverhalt klar und leicht verständlich oder umständlich („verkompliziert“) und mißverständlich beschrieben werden. Kurze Sätze mit prägnanten Begriffen, verknüpft durch eine systematische Gliederung, helfen einen Sachverhalt besser, klarer und schneller zu vermitteln als lange Sätze mit stark verschachtelten Bezügen. In jeder Programmiersprache, also beispielsweise auch in Java^{TM}, steht man vor einem ähnlichen Formulierungsproblem. Es können leicht durchschaubare (transparente) oder undurchschaubare bzw. nur sehr schwer durchschaubare Texte notiert werden. Es geht daher um die Wahl geeigneter Begriffe (Sprachkonstrukte) und um ihre Kombination zu einem transparenten Text, der den Sachverhalt (die Problemlösung) so einfach wie möglich vermittelt.

Assembler

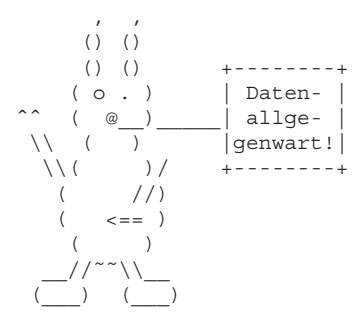
1.3.3 Organisation: Flexibilität & Vereisungseffekt

Das WI-Element Organisation umfaßt gegenläufige Einflüsse auf ein Unternehmen: Einerseits bewirkt die Allverfügbarkeit von Daten mehr Flexibilität andererseits verfestigt der Computereinsatz den IST-Zustand.

Allverfügbarkeit von Daten

Einfachheit

Mit Hilfe von Computernetzen können die unternehmensrelevanten Daten überall dort bereitgestellt werden, wo sie gebraucht werden. Diese Ubiquität[48] (≈Allgegenwart) der Daten ermöglicht mehr Flexibilität in den Arbeitsabläufen und damit auch flexibele Organisationsformen. In der papierbezogenen Arbeitswelt waren die Daten nur dort verfügbar, wo sich das Schriftstück gerade befand. Beispielsweise konnte über die Daten eines Lieferscheins nur derjenige Mitarbeiter Auskunft gegeben, der den Lieferschein vorliegen hatte. Auch mit einer größeren Anzahl von Fotokopien oder Durchschriften des Lieferscheins konnte nur ein relativ begrenzter Personenkreis informiert werden.



Auf der Basis von Computernetzen können die Daten allen Berechtigten (Mitarbeitern, Geschäftspartnern, Behörden, usw.) verfügbar gemacht werden. Die Zusammenarbeit läßt sich unabhängig vom Entstehungsort oder Speicherort der Daten organisieren, da die Daten über Computernetze überall erreichbar sind. Damit ergeben sich weitreichende Gestaltungsmöglichkeiten für die Aufbauorganisation und die Ablauforganisation eines Unternehmens. Zwei Aspekte rücken dabei in den Mittelpunkt:

- **unternehmensübergreifende Wirkungskette**
  Die Datenverarbeitung bezieht sich nicht nur auf ein Unternehmen, sondern wird entlang der Produktions- und Vertriebsprozesse, das heißt entlang der Wirkungskette, organisiert. Holzschnitt-

Ubiquität

[48] Ein Ubiquist ist eine auf der ganzen Erdkugel verbreitete Pflanzen- oder Tierart. Ubiquität ist die Allgegenwart (Gottes).

artig formuliert: Vom ersten Kundenwunsch über alle Produktions- und Dienstleistungsprozesse bis hin zur Entsorgung werden die relevanten Daten weitergereicht. Parallel zu allen Arbeitsvorgängen werden die anfallenden Datenströme organisiert. Die Abhängigkeiten und Rechtsformen der beteiligten Organisationseinheiten spielen bei dieser Zusammenarbeit eine geringere Rolle als die Datenverfügbarkeit. Es läßt sich daher eine Organisationseinheit für die Zusammenarbeit um die Daten, also „virtuell", konstruieren. Diese Einheit existiert nicht „real"[49], sondern nur im Computernetz.

- **Dezentralisierung**
  Da quasi jedes Handeln eines Mitarbeiter nachvollziehbare Datenspuren hinterläßt, kann die jeweils verantwortliche Führungsebene Entscheidungskompetenz an ihn übertragen. Die Führungsfunktion läßt auf der Datenbasis wesentlich einfacher wahrnehmen.

Festgeschriebene Nutzungssituationen

Der Computer ist eine potentiell universelle Maschine. Software gilt als unbegrenzt änderbar und daher auch als unbegrenzt flexibel. Für komplexe Anwendungen trifft dies faktisch jedoch nur in der ersten Entwurfsphase zu. Mit der Inbetriebnahme einer Anwendungssoftware werden die im Entwurf vorhergesehenen Nutzungssituationen festgeschrieben. Die betroffene Organisation verliert einen Teil ihrer Flexibilität („Vereisungseffekt"). Im betrieblichen Alltag kennt man die Entschuldigung: „Können wir nicht machen, der Computer will es nur so.".

Betriebswirtschaftliche Standardsoftware

Das Problem eine marktkonforme Flexibilität sicherzustellen verschärft sich beim Einsatz einer betriebswirtschaftlichen Standardsoftware, insbesondere wenn die Softwareunterstützung alle Bereiche und sämtliche Geschäftsprozesse eines Unternehmens umfaßt. Die Komplexität eines solchen integrierenden Systems und die Abhängigkeit von Entscheidungen des Softwareherstellers verzögern die Anpassungsgeschwindig-

[49] Zum Beispiel gibt es keine eigenen Büroräume.

keit. Damit sollen hier die Vorteile einer solchen Standarsoftware keinesfalls unterbewertet werden. Das betriebswirtschaftliche Wissen und die mannigfaltigen Einsatzerfahrungen des Softwareherstellers fließen einem Unternehmen beim Einsatz direkt zu. Als Beispiel sei hier die Marktführersoftware R/3[50] der SAP[51] AG, Walldorf genannt. Beim System R/3 erhält der Lizenznehmer nicht nur die Software, sondern auch erprobte Hinweise zur Einführung und zum Betrieb. Beispielsweise läßt sich das Vorgehensmodell zur R/3-Einführung, notiert in XML, wie folgt skizzieren:

Wirkungskette

R/3-Vorgehensmodell skizziert als XML-Datei `referenzSAP.xml`

```
 1  <?xml version="1.0" encoding="ISO-8859-1"?>
 2  <!-- Bonin    17-Dec-1999                -->
 3  <!DOCTYPE VORGEHENSMODELL [
 4    <!ELEMENT VORGEHENSMODELL (
 5      ORGANISATIONSPHASE, KONZEPTIONSPHASE,
 6      DETAILLIERUNGSPHASE, REALISIERUNGSPHASE,
 7      PRODUKTIONSVORBEREITUNGSPHASE,
 8      PRODUKTIVPHASE)>
 9    <!ELEMENT ORGANISATIONSPHASE (#PCDATA)>
10    <!ELEMENT KONZEPTIONSPHASE (#PCDATA)>
11    <!ELEMENT DETAILLIERUNGSPHASE (#PCDATA)>
12    <!ELEMENT REALISIERUNGSPHASE (#PCDATA)>
13    <!ELEMENT PRODUKTIONSVORBEREITUNGSPHASE (#PCDATA)>
14    <!ELEMENT PRODUKTIVPHASE (#PCDATA)>
15  ]>
16  <VORGEHENSMODELL>
17    <ORGANISATIONSPHASE>
18      Projektbildung,
19      Projektschulung,
20      Kapazitätenplaung und
21      Terminplanung.
```

[50] Standardpaket R/3 (R ≡ Real Time) wurde vom *Mainframe*-orientierten Vorgängersystem R/2 ausgehend entwickelt. Näheres zu R/3 ↪`http://www.sap-ag.de/germany/products/r3/index.htm` Zugriff 17-Dec-1999.

[51] Die SAP AG ist eines der größten Softwarehäuser der Welt. Sie wurde 1972 von fünf ehemaligen IBM-Ingenieuren gegründet. SAP beschäftigt 1999 mehr als 22.000 Mitarbeiter in mehr als 50 Ländern. Näheres zum Unternehmen SAP ↪`http://www.sap-ag.de/company/index.htm` Zugriff 17-Dec-1999.

```
22    </ORGANISATIONSPHASE>
23    <KONZEPTIONSPHASE>
24      Erarbeiten der SOLL-Konzeption durch
25      Spezifizieren von Geschäftsprozessen,
26      Funktionen und Schnittstellen.
27    </KONZEPTIONSPHASE>
28    <DETAILLIERUNGSPHASE>
29      Verfeinerung der Konzeptergebnisse,
30      Abbildung auf die R/3-Optionen.
31    </DETAILLIERUNGSPHASE>
32    <REALISIERUNGSPHASE>
33      Datenmodellierung,
34      Customizing,
35      Interfacegestaltung,
36      Integration von bestehenden Systemen und
37      Tests.
38    </REALISIERUNGSPHASE>
39    <PRODUKTIONSVORBEREITUNGSPHASE>
40      Datenübernahme aus Altsystemen,
41      Anwenderschulung und
42      Dokumentationserstellung.
43    </PRODUKTIONSVORBEREITUNGSPHASE>
44    <PRODUKTIVPHASE>
45      Systemstart,
46      Benutzerbetreuung und
47      Aktzeptanzverbesserung.
48    </PRODUKTIVPHASE>
49  </VORGEHENSMODELL>
50  <!-- End of object referenzSAP.xml              -->
51
```

Outsourcing — Integration von IT-Experten

Um sich im Unternehmen auf das eigentliche Geschäft (die Kernkompetenz) konzentrieren zu können werden häufig Aufgaben der Informationsverabeitung und Kommunikation organisatorisch ausgegliedert und IT-Experten übertragen. Dieses sogenannte *Outsourcing* von Teilen der IT an einen externen oder zum Konzern gehörenden IT-Dienstleiter kommt in vielfältigen Formen vor, zum Beispiel als:

- **Application Hosting** (AH)
  Der Betrieb und die Wartung einer Anwendung erfolgen durch den Dienstleister. Ein hoher Anteil der in der Bundesrepublik Deutsch-

land abgeschlossenen Verträge zum SAP-R/3-Outscourcing fällt in diese Kategorie.

- **Application Service Providing** (ASP)
  Beim sogenannten Mietsoftwaremodell stellt der Dienstleiter (ASP-Provider) eine Anwendung zur Verfügung und erhält dafür nutzungsabhängige Gebühren. Der Auftraggeber erwirbt weder IT-Infrastruktur noch Softwarelizenzen.

- **Applikation-Management** (AM)
  Neben dem Betrieb und die Wartung geht auch die Aufgabe der Weiterentwicklung auf den Dienstleister über. Dieser betreut eine Anwendung des Auftragebers während der gesamten Lebenszeit.

- **Business Process Outsourcing** (BPO)
  Ein kompletter Geschäftsprozess wird an den Diestleister ausgelagert. Üblich ist beispielsweise die Auslagerung der Lohn- und Gehaltsabrechnung.

1.3.4 Nutzer und Betroffene: Interessengeprägte Rollen

Unstrittig muß sich die WI mit den Rollen und Interessen der Nutzer (*user*) und Betroffenen ihrer Produkte auseinandersetzen. Schon bei einem einzelnen Informationssystem sind vielfältige zum Teil konkurrierende Interesen zu berücksichtigen. Mit einer Analogie zur Architektur[52] lassen sich diese unterschiedlichen Interessenschwerpunkte verdeutlichen. In der XML-Datei `infosys.xml` (↪Seite 68) ist die Sicht aus der WI als Attribut `WISICHT` zum Element `ROLLE` definiert. Die Analogie zur Architektur ist als Attribut `BAUSICHT` notiert. Dabei „hat“ ein Informationssystem (`ELEMENT INFOSYSTEM`) verschiedene Rollen. Das `ELEMENT ROLLE` ist mit verschiedenen Interessenschwerpunkten (`ELEMENT INTERESSE`) verknüpft.

R/3

`infosys.xml`

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <!-- Bonin 11-Nov-1999                    -->
3  <!DOCTYPE INFOSYSTEM [
```

[52] ↪dazu zum Beispiel [Bo88].

```
<!ELEMENT INFOSYSTEM (ROLLE+)>
<!ELEMENT ROLLE (INTERESSE+)>
  <!ATTLIST ROLLE WISICHT ID #REQUIRED
                  BAUSICHT CDATA #REQUIRED>
<!ELEMENT INTERESSE (#PCDATA)>
]>
<INFOSYSTEM>
  <ROLLE WISICHT="Nutzer" BAUSICHT="Bewohner">
    <INTERESSE>
      Erfüllung der fachlichen Anforderungen zuverlässig und
      in hinreichender Qualität.
    </INTERESSE>
    <INTERESSE>
      Softwareergonomie, Verstehbarkeit und Lernbarkeit.
    </INTERESSE>
  </ROLLE>
  <ROLLE WISICHT="Betroffene" BAUSICHT="Nachbarn">
    <INTERESSE>
      Fragen der Schadensbegrenzung, das heißt Einflußnahme
      auf die Ausschaltung von negativen Wirkungen,
      wirksame Auflagen.
    </INTERESSE>
    <INTERESSE>
      Schnittstellen zu (eigenen) Informationssystemen.
    </INTERESSE>
  </ROLLE>
  <ROLLE WISICHT="Auftraggeber" BAUSICHT="Bauherr">
    <INTERESSE>
      Optimale Erreichung der Ziel- und Wunschvorstellungen.
    </INTERESSE>
    <INTERESSE>
      Günstige Nutzen- und Kostenrelation.
    </INTERESSE>
  </ROLLE>
  <ROLLE WISICHT="Systemdesigner" BAUSICHT="Architekt">
    <INTERESSE>
      Konzeptions- und Entwurfsfragen, um das Konglomerat
      an gewünschten, erhofften Softwareleistungen
      ("Benutzermaschine") auf das konkrete
      (das heißt vorhandene oder zu beschaffende)
      Computernetzwerk ("Basismaschine") abbilden zu können.
    </INTERESSE>
  </ROLLE>
  <ROLLE WISICHT="Programmierer" BAUSICHT="Bauunternehmer">
    <INTERESSE>
      Verständlichkeits- und Interpretationsfragen,
```

```
50        um durchschaubare und zuverlässige Programme
51        erstellen zu können.
52      </INTERESSE>
53    </ROLLE>
54    <ROLLE WISICHT="Betreiber" BAUSICHT="Hausmeister">
55      <INTERESSE>
56        Leichte Bedienbarkeit und Zuverlässigkeit.
57      </INTERESSE>
58    </ROLLE>
59    <ROLLE WISICHT="Wartungsdienst" BAUSICHT="Handwerker">
60      <INTERESSE>
61        Änderungs- und Anpassungsfreundlichkeit, insbesondere die
62        Nachvollziehbarkeit der Ablaufstrukturen.
63      </INTERESSE>
64    </ROLLE>
65    <ROLLE WISICHT="Dritte" BAUSICHT="Baubehörden">
66      <INTERESSE>
67        Erfüllung von Gesetzen, Vorschriften, Standards und Normen.
68      </INTERESSE>
69    </ROLLE>
70  </INFOSYSTEM>
71  <!-- End of object infosys.xml                -->
72
```

Rolle

Benutzer-maschine

Sprach-problem

Auf dem Weg von der „Benutzermaschine“ (≈ Requirements & Hoffnungen) zur „Basismaschine“ (≈ Software & Computernetzwerk) sind daher unterschiedliche, zum Teil sogar gegenläufige Interessen zu meistern. Dabei gilt es ein schwieriges Sprachproblem zu lösen. Während die Anforderungen üblicherweise in der Terminologie des Anwendungsfeldes formuliert werden, muß die Spezifikation der Basismaschine in einer Programmiersprache, also einer formalen Sprache, geschehen. Die weitgehend unscharfen, auslegungsfähigen Formulierungen auf der Ebene der Benutzermaschine müssen in die notwendige Eindeutigkeit für die Basismaschine „übertragen“ werden. Dazu sind beim Programmieren laufend zusätzliche Annahmen (Festlegungen) zu treffen, wobei häufig unklar bleibt, welches Interesse dabei mehr Gewicht bekommt.

Für die Nutzer und die Betroffenen hat sich die Situation gegenüber den WI-Anfängen der 60iger Jahre nicht nur entschieden durch den Zuwachs an Gestaltungsoptionen aufgrund permanent leistungsfähigerer Hard- und Software verändert, sondern ebenso durch eine sich wandelnde Einstellung zur Informatik. Zu Beginn des Einsatzes wurden Computer primär als zusätzliches (Hilfs-)Mittel für eine verbesserte Erledigung von Aufgaben angesehen und entsprechend präsentiert und ver-

marktet.[53]

Mit den Erfolgen in komplexen Aufgabenfeldern, wie zum Beispiel auf dem Gebiet der Rentenberechnung, wo faktisch heute allein Computer noch in der Lage sind, aus dem Vorschriftendickicht eine formal korrekte Rentenauskunft zusammenzufügen, wird fühlbar, daß die These: „Computertechnik ist nur lebenserleichterndes Werkzeug“, nicht mehr ausreicht, um erkennbare Fakten und Symptome zu erklären.

Mit wachsender Computerunterstützung wird deutlich, daß das lebenserleichternde (Hilfs-)Mittel gravierende Veränderungen aller Einschätzungen, Beurteilungen und Denkprozesse bewirkt. Damit bekommt die ursprüngliche Einschätzung aus der Zeit der Informatikanfänge:„Computer ist ein Hilfsmittel“ vermehrt die Bedeutung im Sinne, wie man die Fähigkeiten eines Menschen, lesen und schreiben zu können, als Hilfsmittel für ihn auffassen könnte.

Ein Nutzer, der einem Netzwerk aus Hochleistungscomputern gegenüber steht, empfindet nicht primär die traditionelle Hilfsmittelthese in der Art, daß es sich um ein wirksames Werkzeug handelt. In seiner ganzen Erfahrungswelt gibt es keinen anderen Vergleich als mit dem Menschen, schon allein aufgrund der Vielfalt des Verhaltens, der symbolischen Ausdruckskraft und des „Gedächtnisses“. Spontan redet der Nutzer die Maschine wie einen Mitmenschen an. Sätze wie: „Er macht . . . oder macht nicht . . . “ sind allgemein üblich.

Basismaschine

Das permanent wachsende Leistungspotentials erfordert einen Wechsel zum Paradigma[54] Mensch-Maschine-Kooperationsmodell[55] verknüpft mit einem veränderten Selbstverständnis des Menschen. Das Mensch-Maschine-Kooperationsmodell betont eine Symbiose[56] zwischen Menschen und den von ihnen gestalteten Informationssystemen, und zwar im Hinblick auf Gewinnung zusätzlicher Fähigkeiten und Optionen. Der Begriff „Kooperation“ vermittelt einen umfassenderen Aspekt als der eines Mensch-Maschine-Dialogs (Mensch-Maschine-Kommunikation),

„Er macht . . .“

[53] Klassisches, heute auch noch formuliertes Motto: „Computer ist gleich ein besserer Rechenschieber“.

[54] Ein Paradigma ist ein von der Fachwelt als Grundlage der weiteren Arbeiten genutztes Erklärungsmodell. Es entsteht, weil es beim Problemlösen erfolgreicher ist (zu sein erscheint) als andere Ansätze.

[55] ↪dazu [Bo83]

[56] Im Sinne einer Systemökologie, wobei die Computersysteme eine wesentliche Veränderung der Umwelt bewirken.

bestehend in einem Austausch von Informationen, beispielsweise mit Vorzugsrichtung vom Menschen zur Maschine („klassischer Datenerfassungsplatz“) oder umgekehrt („klassischer Auskunftsplatz“).

Mit dieser Kooperationsperspektive sollen die vielfältigen Gestaltungschancen und auch die erheblichen Gestaltungsrisiken für die symbiotische Zusammenarbeit zwischen Menschen und Computern in den Mittelpunkt der Denkwelt der WI gerückt werden.

Klar ist, zukünftig werden wir auf der Basis der weiteren Nutzung und Entwicklung das „Ding Computer“ neu begreifen müssen. Klar ist aber auch, bisher begreifen wir ihn üblicherweise als Werkzeug, als Automat und als *instrumentales Medium*[57].

1.4 Übungen

1.4.1 These: Informationssystem ≫ Hilfsmittel

Erläutern Sie warum ein komplexes Informationssystem nicht hinreichend mit der sogenannten Hilfsmittelthese erklärbar ist.

1.4.2 Akronyme auflösen

Erläutern Sie die folgenden Akronyme: DBMS, GUI, LAN, WAN, XML, DTD, `#PCDATA`, HTML und UML.

1.4.3 Was ist eine Plattform?

Skizzieren Sie den Begriff „Plattform“ in Form einer XML-Notation.

1.4.4 DTD interpretieren

Interpretieren Sie die folgende *Document Type Definition* und notieren Sie Ihr Ergebnis in Form eines leicht verständlichen Textes.

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <!-- Bonin    09-Sep-1999                -->
3  <!ELEMENT LOGISTIK
4    (WIRTSCHAFTLICHES-HANDELN)>
5  <!ELEMENT WIRTSCHAFTLICHES-HANDELN
```

[57] Begriff stammt von Heidi Schelhowe und Frieder Nake ↪ [Na03], S. 219.

```
 6    (EINSATZ*, LAGERUNG+, PRODUKT+)>
 7  <!ELEMENT EINSATZ (PROZESS+)>
 8  <!ELEMENT LAGERUNG (PROZESS+)>
 9  <!ELEMENT PROZESS
10    (PLANUNG+, DURCHFUEHRUNG+,
11    UEBERWACHUNG+, KONTROLLE+)>
12  <!ELEMENT PRODUKT
13    (FERTIGPRODUKT | HALBFERTIGPRODUKT
14    | ROHMATERIAL)>
15  <!ELEMENT PLANUNG (#PCDATA)>
16  <!ELEMENT DURCHFUEHRUNG (#PCDATA)>
17  <!ELEMENT UEBERWACHUNG (#PCDATA)>
18  <!ELEMENT KONTROLLE (#PCDATA)>
19  <!ELEMENT FERTIGPRODUKT (#PCDATA)>
20  <!ELEMENT HALBFERTIGPRODUKT (#PCDATA)>
21  <!ELEMENT ROHMATERIAL (#PCDATA)>
22  <!-- End of object logistik.dtd            -->
23
```

1.4.5 „EVA“-Modell erklären

Erläutern Sie die Phasen des „EVA“-Modells. Was versteht man unter „Verarbeitung“ von Daten?

Kapitel 2

Abbildung von Daten

Auf einem Computer zeigt sich die „reale Welt“ in Form von *Daten*[1]. Gegenstände und ihre Beziehungen untereinander werden durch Daten repräsentiert (abgebildet). Das aktive Veränderung von Daten erfolgt durch das Anwenden von Algorithmen. Ein Algorithmus selbst wird ebenfalls in Form von Daten repräsentiert. Daten sind daher sowohl *aktive* und *passive* Teile des Systems (↪Tabelle 2.1 auf Seite 76).

Wegweiser

Das Kapitel „Abbildung von Daten“ erläutert:

- Elementarteilchen des *Computings*, insbesondere die Begriffe Bit, Byte, Wort und Wahrheitswert,
  ↪ Seite 76 . . .
- die Darstellung von Zahlen,
  ↪ Seite 85 . . .
- den Datentyp und seine Prüfung,
  ↪ Seite 88 . . .

[1] Dabei bedeutet der Begriff „Daten“ etwas „Gegebenes“ (vom Lateinischen *datum*). Der Begriff „Daten“ ist Plural. Der Singular heißt „Datum“ und wird häufiger als „Datenelement“ bezeichnet.

	Abbildung	
Computerebene	$\Longleftrightarrow$	**Reale Welt**
	passive Teil	
Daten	$\Longleftrightarrow$	Gegenstände
Attribute von Daten	$\Longleftrightarrow$	Eigenschaften von Gegenständen
$\equiv$ (selbst) wieder Daten		
Relationen zwischen Daten	$\Longleftrightarrow$	Beziehungen zwischen
$\equiv$ (selbst) Daten		Gegenständen
	aktive Teil	
Algorithmen	$\Longleftrightarrow$	Handlungen
$\equiv$ (selbst) Daten		

Tabelle 2.1: Abbilung: Reale Welt $\Longleftrightarrow$ Daten

- Metadaten als Daten über Daten und
  ↪ Seite 105 . . .
- das Konzept *Data Warehouse* als eine charakteristische Lösung für große Mengen von betriebswirtschaftlichen Daten.
  ↪ Seite 118 . . .

2.1 Bit, Byte, Wort und Wahrheitswert

Auf dem Computer ist *Rechnen* ein Erzeugen, Ändern und/oder „Löschen“ von Daten. Üblicherweise verbindet man mit Rechnen den Umgang mit Symbolen, insbesondere mit denen, die Zahlen darstellen. Bei den Zahlen sind die „natürlichen Zahlen“ die sogenannten Elementarteilchen (*primitives*). Auf sie stützt sich eine ganze Zahlenhierarchie ab (↪Tabelle A.3 auf Seite 259). Bei den Daten ist es das „Bit“ (*binary digit*). Es steht für die Binärziffern „0“ oder „1“. Auch das Bit ist die Basis für weitere elementare Einheiten. So bildet eine Folge von acht

$0 \vee 1$

Bits die Einheit *Byte*[2] (↪Tabelle 2.2 auf Seite 78) und die Folge von 2 Bytes ein 16-Bit-Wort. Ein Halbbyte wird auch als „Nibble“ bezeichnet. In XML läßt sich daher für die Abbildung eines Wortes aus lauter 0-Bits wie folgt notieren:

```
 1  <?xml version="1.0" encoding="ISO-8859-1"?>
 2  <!-- Bonin: 12-Nov-1999                  -->
 3  <!DOCTYPE WORT [
 4  <!ELEMENT WORT (BYTE, BYTE)>
 5  <!ELEMENT BYTE (NIBBLE, NIBBLE)>
 6  <!ELEMENT NIBBLE (BIT, BIT, BIT, BIT)>
 7  <!ELEMENT BIT EMPTY>
 8    <!ATTLIST BIT WERT (0 | 1) #REQUIRED>
 9  ]>
10  <WORT>
11    <BYTE>
12      <NIBBLE>
13        <BIT WERT="0" />
14        <BIT WERT="0" />
15        <BIT WERT="0" />
16        <BIT WERT="0" />
17      </NIBBLE>
18      <NIBBLE>
19        <BIT WERT="0" />
20        <BIT WERT="0" />
21        <BIT WERT="0" />
22        <BIT WERT="0" />
23      </NIBBLE>
24    </BYTE>
25    <BYTE>
26      <NIBBLE>
27        <BIT WERT="0" />
28        <BIT WERT="0" />
29        <BIT WERT="0" />
30        <BIT WERT="0" />
31      </NIBBLE>
32      <NIBBLE>
33        <BIT WERT="0" />
34        <BIT WERT="0" />
35        <BIT WERT="0" />
36        <BIT WERT="0" />
```

[2] Englisches Kunstwort

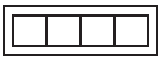
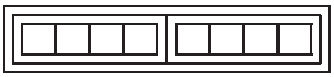
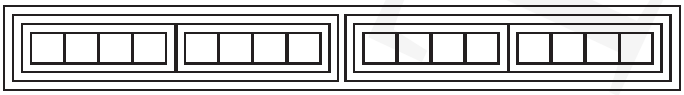
Bit	
Halbbyte (*nibble*)	
Byte	
16-Bit-Wort	
1 KiloByte (KB)	= 1024 Byte = 2^{10} Byte $\approx 10^3$ Byte
1 MegaByte (MB)	= 1048576 Byte (= 1024 KB) = 2^{20} Byte $\approx 10^6$ Byte
1 GigaByte (GB)	= 1073741824 Byte (= 1024 MB) = 2^{30} Byte $\approx 10^9$ Byte
1 TeraByte (TB)	= 1099511627776 Byte (= 1024 GB) = 2^{40} Byte $\approx 10^{12}$ Byte
	weiter ↪Tabelle 2.3 S. 79

Tabelle 2.2: Bit, Byte, Wort

```
37       </NIBBLE>
38     </BYTE>
39   </WORT>
40   <!-- End of object wort.xml                  -->
41
```

Ein Zeichen z_i ist in einem Computer ein eindeutiges elektrisches Signal.[3] Ein (Signal-)Alphabet Z ist eine endliche Menge von Zeichen $Z = \{z_1, z_2, \ldots, z_n\}$. Es gibt viele Alphabete, zum Beispiel:

- Alphabet der Wochentage:
  {Sonntag, Montag, Dienstag, Mittwoch, Donnerstag, Freitag, Sonnabend}

[3] „Wir sagen: Zeichen werden zu Signalen beim Übergang in den Computer hinein. Und meinen mit *Signal* das seiner wesentlichen Eigenschaft, der Interpretierbarkeit nämlich, entkleidete Zeichen.. . . Was geschieht, grenzt an Wunder: Signalprozesse sind mit Zeichenprozessen so gekoppelt, dass das Ganze für uns einen Sinn ergibt (den wir naturgemäß erst herstellen). . . . Die Kopplung maschineller, expilziter, vorherbestimmter Signalprozesse mit menschlichen, impliziten, unendlichen offenen Zeichenprozessen ist das Geheimnis, die Aufgabe und der Erfolg der Mensch-Computer-Interaktion. Die Benutzungsoberflächen sind die Orte und Mittel dieser Kopplung.“ ↪ [Na03] S. 222.

Einheit	kurz	Wert	
Byte	B	8	Bits
KiloByte	KB	1024	Byte
MegaByte	MB	1024	KiloByte
GigaByte	GB	1024	MegaByte
TeraByte	TB	1024	GigaByte
PetaByte	PB	1024	TeraByte
ExaByte	EB	1024	PetaByte
ZettaByte	ZB	1024	ExaByte
YottaByte	YB	1024	ZettaByte

Tabelle 2.3: Einheiten für ein großes Datenvolumen

- Alphabet der Jahreszeiten:
  {Frühling, Sommer, Herbst, Winter}
- Alphabet der Dezimalsziffern
  {0,1,2,3,4,5,6,7,8,9}

Eine Zeichenkette (*String*):

$$s = z_1 \circ z_2 \circ \ldots \circ z_n$$

ist eine Folge, deren Zeichen z_i nicht notwendig verschieden sein müssen. Der Verkettungsoperator „$\circ$“ , auch *Konkatenation* genannt, kann weggelassen werden, wenn durch diese Kurzschreibweise der Zusammenhang weiterhin eindeutig bleibt. **Konkatenation**

$$s = z_1 z_2 \ldots z_n$$

Die Zeichen können als Signalfolge in kodierter Form vorliegen. Die Signale werden in der digitalten Schaltungstechnik als **high** und **low** oder **null** und **eins** bezeichnet.

2.1.1 Codierung

Das im Computer zu „verarbeitende“ Alphabet wird als Folge binärer Signale definiert. Diese Abbildung (Umsetzung) heißt Codierung.[4] Eine

[4] Näheres zur Codierung beispielsweise in ↪[Coy88].

Codierung c definiert die Zuordung einer endlichen Menge von Zeichen eines Alphabets A in eine Signalfolge über die unterliegende Signalmenge S.

Morse

Einen für die Nachrichtenübertragung bedeutsamen Code hat der Amerikaner Samuel Morse[5] (1791–1872) entwickelt. Der Morsecode basiert auf einer Abbildung in die Menge $M = \{\bullet, -\}$:

$$c_{Morse} : A \rightarrow \{\bullet, -\}^n$$

Der Morsecode ist nicht längenkonstant. Die Zahl n schwangt bei Buchstaben zwischen 1 und 4. Die Ziffern sind 5 Zeichen lang (↪Tabelle A.4 auf Seite 260). Bei der Übertragung wird zur Abgrenzung des einzelnen Zeichens und einer Zeichenfolge eine Pause von einer bestimmten Länge gemacht. Damit gehört auch das Leerzeichen „␣“ zur Signalmenge $S = \{\bullet, -, \sqcup\}$.

Die Darstellung von Zahlen fußt auf der Zerlegung einer natürlichen Zahl n nach Potenzen einer Basis B (↪Tabelle 2.4 auf Seite 81). Gewöhnlich wird die *Stellenwertschreibweise* auf der Basis 10 genutzt. Sie leitet sich aus der *Potenzschreibweise* ab; zum Beispiel:

$$5 * 10^2 + 1 * 10^1 + 7 * 10^0 \equiv 517$$

BCD

In der zweiwertigen (digitalen) „Welt“ ist die Basis $B = 2$. Die Codierung der Dezimalzahlen in Einheiten der festen Länge von 4 Bits ($\equiv$ Halbbyte) heißt BCD-Code (*Binary Coded Decimal*) (↪Tabelle A.5 auf Seite 261). Unter der Hinzunahme eines weiteren Halbbytes hat IBM für ihre Großrechner (*Mainframes*) daraus die Codierung EBCDIC (*Extended Binary Coded Decimal Interchange Code*) entwickelt. Im Bereich der Personalcomputer und Workstations ist heute die ASCII-Codierung (*American Standard Code for Information Interchange*) verbreitet (↪Tabelle A.6 auf Seite 262 und Tabelle A.7 auf Seite 263). Der ASCII-Code wird trotz seiner 7 Bit-Länge (in ISO-Norm) häufig byteweise „verarbeitet“. Das achte Bit ist dann meist eine Null. Leider gibt es unterschiedliche Vereinbarungen über die restlichen 128 Zeichen (von 1000 0000 bis 1111 1111).

EBCID

[5] Die erste Telegraphenlinie wurde am 27.05.1843 zwischen Washington und Baltimore eröffnet.

Zerlegen einer Zahl n nach Potenzen der Basis B:

$$n = a_k * B^k + a_{k-1} * B^{k-1} + \ldots + a_1 * B^1 + a_0 * B^0$$

wobei:
die Koeffizienten $a_k, \ldots, a_0$ der Basispotenzen $B^k, \ldots, B^0$
aus der B-elementigen Ziffernmengen
$Z_B = \{0, 1, 2, \ldots, B-1\}$ stammen müssen.

Beispiel: „4711“

$$\begin{array}{rll} n = & 1*2^{12}+0*2^{11}+0*2^{10}+1*2^9+0*2^8 & \\ & +0*2^7+1*2^6+1*2^5+0*2^4 & \\ & +0*2^3+1*2^2+1*2^1+1*2^0 & = (1001001100111)_2 \\ n = & 1*8^4+1*8^3+1*8^2+4*8^1+7*8^0 & = (11147)_8 \\ n = & 4*10^3+7*10^2+1*10^1+1*10^0 & = (4711)_{10} \end{array}$$

Tabelle 2.4: Zahlendarstellung

2.1.2 Digitalisierung

Physikalische Größen sind im Regelfall stetig veränderlich und können daher unendlich viele Werte annehmen. Zum Beispiel gilt dies für die Größen „Zeit“ oder „Temperatur“. Sind solche Größen durch binärcodierte Daten darzustellen, dann müssen zunächst die unendlich vielen Werte durch endlich viele ersetzt werden. Als *Diskretisierung*[6] oder *Rasterung* bezeichnet man die Ersetzung eines unendliche Wertebereichs durch einen endlichen. Die Diskretisierung ist prinzipiell ungenau und verfälschend. Ihr Fehler kann jedoch durch die gewählte endliche Wertemenge unter die Fehlertoleranzgrenze gedrückt werden.

diskret

Die Codierung der endlich vielen Werte des diskretisierten Wertebereiches als Zahlen nennt man *Digitalisierung*. „Digital“[7] bedeuted ziffernhaft. Unter der Formulierung „Daten sind digital repräsentiert“ versteht man (im engeren Sinne) die Darstellung „durch Zahlen“. Im Computer sind alle Daten mit Hilfe eines Binärcodes digital — also als Zahlen — dargestellt. Tabelle 2.5 Seite 82 skizziert nochmals die Phasen

digital

[6] *diskret* ≡ unstetig, getrennt — im Sinne der Mathematik
[7] Englisch: *digit* ≡ Ziffer; aus dem Lateinischen: *digitus* ≡ Finger

Analoge Größe $\Longrightarrow$ Bytes		
	analoge Größe:	
unendlich viele Werte	$\longrightarrow$(DISKRETISIERUNG)$\longrightarrow$	endliche viele Werte
	endlich viele Symbole:	
Zeichen aller Art	$\longrightarrow$(DIGITALISIERUNG)$\longrightarrow$	Zahlen
Zahlen	$\longrightarrow$(BINÄRCODIERUNG)$\longrightarrow$	Bytes

Legende:

binär	$\equiv$	Darstellung von Zeichen durch ein zweiwertiges Alphapet, gleichgültig, ob es sich um Ziffern, Buchstaben oder andere Zeichen handelt.
dual	$\equiv$	Darstellung von Zahlen im *dualen Zahlensystem*, dessen Stellen von **rechts nach links** die Wertigkeit $1, 2, 4, 8, \ldots$ haben.

Tabelle 2.5: Skizze: Diskretisierung, Digitalisierung, Binärcodierung

bei unendlich vielen Analogwerten.
Beispiel: Temperaturwerte
Die Tempratur soll in einem Bereich von 0 Grad bis 99 Grad auf 1 Grad Celsius erfaßt werden. Gebraucht werden also 100 unterscheidbare Werte. Für die Binärcodierung benötigt man eine (Gruppen-)Länge n für die gilt: $2^n \geq 100$. Daraus folgt $n = 7$, weil $2^7 = 128$. Soll die Genauigkeit auf $\frac{1}{10}$ Grad gesteigert werden, dann ist $n = 10$ zu wählen, weil $2^{10} = 1024 \geq 1000$.

2.1.3 Beispiel: Bar Code

Ein *Bar Code* (Balkencode, Strichcode) ist eine definierte Strichmarkierung, die aufgrund ihrer Hell-Dunkel-Kontraste mittels optischelektronischer Erfassungsgeräten (*Scanner*) lesbar ist. Üblicherweise ist ein *Bar Code* auf Papier, vor allem Verpackungen und selbstklebenden Etiketten gedruckt.

EAN

Ein Alltagsbeipiel für einen genormten Strichcode ist die im Nahrungsmittelbereich verwendete 13stellige EAN (europaeinheitliche Arti-

13-stellige Artikelnummer			
XX	**XXXXX**	**XXXXX**	**X**
Länderkennzeichen (*flag*) des Produktionslandes (BRD: 40–43)	Betriebsnummer des Produzenten (BRD: bundeseinheitliche Betriebsnummer (BNN), die von der CCG auf Antrag vergeben wird.)	Artikelnummer je Hersteller (BRD: liegt in der Verantwortung des jeweiligen Herstellers)	Prüfziffer

Legende:
CCG ≡ Centrale für Coorganisation GmbH, Moorweg 133, D-50825 Köln (Homepage: ↪`http://www.ccg.de/` Zugriff 29-Dec-1999).

Tabelle 2.6: EAN-Aufbau

kelnummer). Die EAN-Einführung wurde 1977 von allen EG-Staaten vereinbart. Mittlerweile haben sich viele Länder, darunter die USA und Japan, dieser Übereinkunft angeschlossen. Jede handelsübliche Mengen- und Verpackungseinheit erhält vom Hersteller eine eigene EAN. Den EAN-Aufbau zeigt Tabelle 2.6 auf Seite 83.

Das EAN-Symbol besteht aus einer Folge von dunklen Balken unterschiedlicher Breite auf hellem Grund. Die Balken sind parallel und senkrecht zur Klarschriftzeile in **OCR**[8]**-B-Schrift** gedruckt. Die visuell lesbare OCR-B-Schriftzeile dient als Sicherheit, falls das Lesesystem die Balken nicht interpretieren kann, zum Beispiel wegen Verschmutzung. Das EAN-Symbol gliedert sich in zwei Teile mit jeweils 6 Zeichen[9]. Die beiden Teile sind durch ein Mittezeichen und die Randzeichen abgegrenzt. Die beiden Teile werden vom Lesegeräte unterschieden und können sowohl von links nach rechts als auch von rechts nach links gelesen werden. Jede Ziffer ist abgebildet in Form von sieben gleichbreiten Teilen, die entweder dunkel (Balken) oder hell (Zwi-

[8] OCR ≡ optical character recognition – die Schriftformen A und B sind in den DIN-Normen 66008 und 66009 genormt.

[9] Für Artikel mit kleinem Volumen gibt es eine achtstellige EAN – hier ist ein Teil 4 Zeichen lang.

„Mündlich“	COBOL	LISP	Elektronsiche Schaltungen	Boolesche Logik
wahr	`TRUE`	„Alles“ $\neq$ `NIL`	$\geq 2,8$ Volt	1
falsch	`FALSE`	NIL	$\leq 2,2$ Volt	0

Tabelle 2.7: Darstellung der beiden Wahrheitswerte

schenräume) sind.

2.1.4 Logische Ausdrücke

Aussagen $(A_1, A_2, A_3, \ldots)$ sind „Objekte“, denen in eindeutiger Weise ein Wahrheitswert zugeordnet werden kann. Wahrheitswerte sind die Werte *wahr* und *falsch* (↪Tabelle 2.7 auf Seite 84). *Junktoren*[10] verbinden die Aussagen, zum Beispiel: $A_1 \vee A_2$; also A_1 oder A_2.

Boole

Eine Größe, die nur die Werte „1“ (*true*) und „0“ (*false*) annehmen kann, bezeichnet man als *logische Variable* oder zu Ehre des Begründers der formalen Logik George Boole (1815–1864) auch als *boolsche Variable*.

Bewertung

Eine konkrete Zuordung von Wahrheitswerten zu (logischen) Variablen heißt *Bewertung*. Bei mehreren Variablen notiert man die Bewertung β_d^n tabellarisch, wie das folgende Beispiel zeigt:

$$\beta_{25}^8 \equiv \begin{array}{c|c|c|c|c|c|c|c} x_7 & x_6 & x_5 & x_4 & x_3 & x_2 & x_1 & x_0 \\ \hline 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{array}$$

Die Werteverteilung unter dem Strich wird als Dualzahl interpretiert und ihr Dezimalwert d ermittelt. Die hochgestellte Zahl n gibt die Anzahl der Variablen an.

Ein logischer Ausdruck kann folgende Komponenten haben:

- die logischen Konstanten „0“ und „1“,

[10] Man bezeichnet daher auch die *Aussagenlogik* als *Junktorenlogik*.

- logischen Variablen,
- Junktoren (↪Tablle A.8 auf Seite 264) und
- Klammern

Ein logischer Ausdruck kann mit Hilfe des Vergleiches aller Werte und/oder der Umformungsregeln (↪Tabelle A.9 auf Seite 265) vereinfacht werden; zum Beispiel:

Wertevergleich: $(a \wedge \overline{b}) \vee (\overline{a} \wedge b) \equiv a \oplus b$

a	b	$a \oplus b$	$(a \wedge \overline{b}) \vee (\overline{a} \wedge b)$
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Umformungsregel: $\overline{(a \vee b) \wedge \overline{c}} \equiv (\overline{a} \wedge \overline{b}) \vee c$

2.2 Zahlendarstellung

Intern werden Zahlen binärcodiert mit fester Wortläng n (zum Beispiel 16, 32 oder 64 Bit) dargestellt. Ein Wort kann nur einen endlichen Zahlenbereich lückenlos darstellen. Bei den natürlichen Zahlen ist das geschlossene Intervall[11] $[0, 2^n - 1]$ abbildbar. Zum Beispiel reicht bei $n = 16$ der lücklose, positve Zahlenbereich von 0 bis 65535. Durch diese Begrenzung kann es bei einer arithmetischen Operation zu einem „Überlaufproblem“ kommen, weil die Obergrenze $2^n - 1$ überschritten wird.

Überlauf

Beispiel: Überlauf (Wortlänge $n = 7$, ohne Vorzeichen):

$a \equiv$ $1001100 \equiv 76_{10}$
$b \equiv$ $\underline{1101100} \equiv 108_{10}$
Übertrag: 1001100

[11] Mit negativen Teilbereich ist es das $[-2^{n-1}, +2^{n-1} - 1]$, wenn „0“ eindeutig dargestellt wird.

$a + b \equiv$ $\underline{\underline{0111000}}$! nur 7 Bit, daher falsch
10111000 ! wäre bei 8 Bit korrekt (184_{10})

Negativen Binärzahlen können unterschiedlich im n-stelligen Wort dargestellt werden. Üblich sind folgende Formen[12]:

- „erste“[13] Bit: $1 \equiv$ *negativ* und $0 \equiv$ *positiv*
  Das Vorzeichen wird im ersten Bit verschlüsselt. Ein Wort mit der Länge n kann dann genausoviele positive Zahlen (2^{n-1}) im Intervall $[+0, \ldots, +2^{n-1} - 1]$ wie negative Zahlen (2^{n-1}) im Intervall $[-2^{n-1} + 1, \ldots, -0]$ darstellen. Nachteilig ist dabei, daß es zwei unterschiedliche Nullwerte gibt, nämlich -0 und $+0$.

- *n-stelliges Einerkomplement* $K_1(x)$:
  Die Menge der darstellbaren Zahlen (2^n) in zwei symmetrische Hälften geteilt und zwar das gilt:

  Bitmuster der negativen Zahl $\equiv (2^n - 1) -$ *positiver Wert der Zahl*

  Es gilt zum Beispiel für „-0“ bei $n = 8$:
  „-0“ $\equiv (2^8 - 1) - 0 = 255_{10}$
  das heißt, alle 8 Bit sind gesetzt (haben den Wert $= 1$).

- *n-stelliges Zweierkomplement* $K_2(x)$:
  Hier gilt die Beziehung:
  Bitmuster der negativen Zahl $\equiv 2^n -$ *positiver Wert der Zahl*

  Die Null ist nun eindeutig im Wort dargestellt, da für ein n-stelliges Wort durch Abschneidung folgender Bitmustervergleich gilt:
  $0_{n-1}, 0_{n-1}, 0_{n-2}, \ldots, 0_0 \equiv 1_n, 0_{n-1}, 0_{n-2}, \ldots, 0_0$

[12] Für eine ausführlichere Diskussion ↪zum Beispiel [Coy88]

[13] Benennen („Adressieren“) kann man *nullbasiert* oder *einsbasiert*. Bei *nullbasierter* Zählweise ist das rechte Bit mit dem Wert 2^0, die sogenannte nullte Position. Bei einer Wortlänge von n hat das linke Bit dann die Position $n - 1$. Bei einer *einsbasierten* Zählweise hat dieses Bit die n-te Position. Salopp nennt man dieses *linke Bit* oft auch das *erste Bit*, weil man es beim Lesen als erstes antrifft.

Die Darstellung im Einerkomplement $K_1(x)$ steht zur Darstellung im Zweierkomplement $K_2(x)$ in folgender Beziehung: $K_2(x) \equiv K_1(x) + 1$. Erzeugen kann man $K_2(x)$, indem man x bitweise negiert und zum Ergebnis 1 addiert, d.h. es inkrementiert. Tabelle A.10 Seite 266 zeigt den Zusammenhang exemplarisch anhand einiger Werte.

2.2.1 Vorzeichenzahlen

Sind zwei Zahlen x und y mit Vorzeichen[14] zu addieren, bedarf es einer Fallunterscheidung nach Vorzeichen und Betrag der Operanden:

1. *gleiches Vorzeichen*:
   Die Addition erfolgt als Addition der positiven Absolutwerte $|x|$ und $|y|$ und das Vorzeichen wird beibehalten.
   Hier gilt: $-x + -y = (-|x|) + (-|y|) = -(|x| + |y|)$

2. *ungleiches Vorzeichen*:
   Bei ungleichem Vorzeichen ist ein Größenvergleich der Absolutwerte $|x|$ und $|y|$ erforderlich, weil die kleiner Zahl von der größeren abzuziehen ist. Hier gilt:

   - für $|x| \geq |y|$
     $|x| + (-|y|) = +(|x| - |y|)$; d. h. mit Ergebnisvorzeichen „+“
   - für $|y| > |x|$
     $|x| + (-|y|) = -(|y| - |x|)$; d. h. mit Ergebnisvorzeichen „−“

Die Subtraktion mit Vorzeichen verläuft analog. Auch sie beruht auf einer Vorzeichenprüfung und dem Größenvergleich von $|x|$ mit $|y|$.

2.2.2 Zweierkomplement

Sind zwei Zweierkomplementzahlen[15] x und y zu addieren, dann werden folgende Fälle unterschieden, wobei das Ergebnis, die Summe s, das „erste“ Bit s_n hat:

[14] In der Darstellung von Tabelle A.10 Seite 266 Spalte II.
[15] In der Darstellung von Tabelle A.10 Seite 266 Spalte IV.

- $x \geq 0$ und $y \geq 0$
  Dieser Fall entspricht der Addition positiver Zahlen. Das Ergebnis ist nicht korrekt, wenn durch einen Überlauf $s_n \neq 0$ würde, weil bei positiven x und y stets gilt: $x_n = y_n = s_n = 0$.

- $x < 0$ und $y < 0$
  Für der Addition zweier negativen Zahlen im Zweierkomplement gilt:
  $(-x) + (-y) \equiv (2^n - x) + (2^n - y) = 2^{n+1} - (x + y)$
  Der Summand 2^{n+1} fällt bei einem Wort der Länge n weg, weil er nur das Bit außerhalb des Wortes betrifft und sonst 0 ist. Es gilt daher $(-x) + (-y) \equiv -(x + y)$, wobei $s_n = 1$ sein muß, weil $x_n = 1$ und $y_n = 1$ definitionsgemäß die Summe negativ sein muß. Sollte $s_n = 0$ sein, liegt ein Überlauf vor.

- *ungleiches Vorzeichen* $x_n \neq y_n$
  Beispiel: $x \geq 0$ also: $x_n = 0$ und $y < 0$ also $y_n = 1$
  Es gilt:
  $x + (-|y|) = 2^n - 2^n + x - |y| = (2^n - |y|) + x - 2^n$
  Addiert man das Zweierkomplement von y zu x dann entsteht ein Ergebnis $s' = -y + x + 2^n$, das um 2^n zu groß ist. Das korrekte Ergebnis liegt im zulässigen Zahlenbereich.

Im Zweierkomplement kann daher eine elektronische Addierschaltung (Wortaddierer) für die Addition und die Subtraktion verwendet werden.

2.2.3 Beispiele mit Wortlänge $n = 8$

Die Tabelle 2.8 auf Seite 89 zeigt die Addition $18+4$. Die Tabelle 2.9 auf Seite 89 zeigt die Addition zweier negativer Zahlen; nämlich $(-33) + (-17)$. Die Tabelle 2.10 auf Seite 90 zeigt die Subtraktion $15 - 33$.

2.3 Datentyp und seine Prüfung

„Datenstrukturen sind
(primär gedankliche) Konstruktionen zur
geordneten Darstellung von Daten . . .“
[Eck1990], Seite 79

In Dezimal-darstellung	$\Longleftrightarrow$ In Zweierkomplement-darstellung
18	00010010
$\underline{+4}$	$\underline{+00000100}$
22	00010110
96	01100000
$\underline{+32}$	$\underline{+00100000}$
128	falsch!! $[1]0000000$
	$s_n = 1$; Überlauf
	128_{10} nicht darstellbar mit $n = 8$!!

Tabelle 2.8: Beispiel: Überlauf bei Addition

In Dezimal-darstellung	$\Longleftrightarrow$ In Zweierkomplement-darstellung
-33	33_{10} als positive Zahl: 00100001
$\underline{-17}$	Bitweise Negation davon: 11011110
-50	Inkrement (+1):
	11011110
	$\underline{+1}$
	11011111
	17_{10} als positive Zahl: 00010001
	Bitweise Negation und Inkrement: 11101111
	11011111
	$\underline{+11101111}$
	$[1]11001110$
	Probe: -50_{10} erzeugen, mit $n = 8$
	$2^8 - 50_{10} = 256_{10} - 50_{10}$
	$= 206_{10}$
	$= 128_{10} + 64_{10} + 8_{10} + 4_{10} + 2_{10}$
	$= 2^7 + 2^6 + 2^3 + 2^2 + 2^1$
	$\equiv 11001110$

Tabelle 2.9: Beispiel: Addition zweier negativer Zahlen

In Dezimal-darstellung	$\Longleftrightarrow$ In Zweierkomplement-darstellung
15	00001111
$\underline{-33}$	$\underline{+11011111}$
-18	11101110
	Probe: -18_{10} erzeugen, mit $n = 8$
	$2^8 - 18_{10} = 238_{10}$
	$= 128_{10} + 64_{10} + 32_{10} + 8_{10} + 4_{10} + 2_{10}$
	$= 2^7 + 2^6 + 2^5 + 2^3 + 2^2 + 2^1$
	$\equiv 11101110$

Tabelle 2.10: Beispiel: Subtraktion

Mit Datenstrukturen[16] bezeichnet man den Aufbau von Wertebereichen aus elementaren Wertebereichen. Verknüpft mit Datenstrukturen sind die auf sie anwendbaren Operationen, das heißt Algorithmen und Datenstrukturen bedingen einander (↪Tabelle A.11 auf Seite 267).

Die Programmiersprachen ermöglichen nur einen Ausschnitt der zugrundeliegenden Wertemengen, etwa $\mathbb{Z}_c = \{n \in \mathbb{Z}: |n| \leq int_{max}\}$ oder $\mathbb{Q}_c = \{x \in \mathbb{Q}: |x| \leq real_{max}\}$, wobei int_{max} und $real_{max}$ abhängig sind von der Programmiersprache und/oder der Hardware.

„Reale Objekte“ kann man mit einer Datenstruktur in Form von *Konstanten* und *Variablen* schaffen. Man spricht dabei von einer *Inkarnation* des entsprechenden Typs. Aus der Speicherperspektive ist eine Variable ein Speicherplatz, der eine bestimmte Anzahl von Wörtern umfaßt. Mathematisch formuliert ist die Variable V ein Konstrukt aus vier Größen, das heißt, ein Quadrupel:

Variable

$$\boxed{V \equiv (T, N, A, W)}$$

mit:

T $\equiv$ Typ
N $\equiv$ symbolischer Name

[16] Häufig wird die Bezeichnung „Datenstruktur“ auch als Synonym für „Datentyp“ verwendet — zum Beispiel Datenstruktur „Baum“.

A $\equiv$ Adresse (des Speicherplatzes)
W $\equiv$ Wert (Inhalt des Speicherplatzes)

In XML notiert:

```
<!ELEMENT VARIABLE (TYP, NAME, ADRESSE, WERT)>
```

Der Name „Variable“ betont, daß sich der Wert W von V zur Laufzeit des Programms ändern kann. Dies führt zu einem wesentlichen Interpretationsunterschied gegenüber der mathematischen Notation:

Assign Statement

$$\boxed{FOO = FOO + 1}$$

Der Wert W der Variablen FOO ist nach Vollzug dieser Anweisung um 1 größer. Die linke Seite vom Gleichheitszeichen verweist auf einen späteren Zeitpunkt als die rechte Seite. Zum besseren Verständnis kann man sich daher folgende Notation vorstellen:

$$\boxed{FOO_{t_1} = FOO_{t_0} + 1}$$

$$\underline{\text{mit}}: t_1 > t_0$$

Es ist also kein Vergleich, sondern eine schrittweise Zuweisung eines neuen Wertes W (*assign statement*). Um Mißverständnissen vorzubeugen haben einige (moderne?) Programmiersprachen ein besonderes Zuweisungssymbol an Stelle des Gleichheitszeichens, zum Beispiel $:=$ oder $\leftarrow$.

Zusammengesetzte Datenstrukturen bestehen aus einfachen Typen (*Primitives*) und/oder enthalten als Bausteine bereits zusammengesetzte Datenstrukturen. In Programmiersprachen sind häufig folgende zusammengestzte Datenstrukturen vordefiniert:
— Reihung (Synonyme: Vektor, Matrix, Feld, Array)
— Verbund (Synonyme: Record, Struktur)
— Graph
— Baum
— Zeigertyp

2.3.1 Hierarchische Datenorganisation: Blockstruktur

Durchschaubarkeit

Bei der Abbildung von Ausschnitten der „realen Welt“ in die „Datenwelt“ geht aufgrund der großen Komplexität, die es zu meistern gilt, schnell die Durchschaubarkeit des Datenmodells verloren. Eine Technik um Transparenz zu sichern, ist die Datenorganisation in Form einer strikten Hierarchie. Die einzelnen Daten werden verschiedenen Hierarchieebenen zugeordnet. Betrachtet man zum Beispiel ein Buch. Es ist üblicherweise strukturiert in Kapitel und Abschnitte. Sichergestellt ist dabei, daß derselbe Abschnitt nur einem Kapitel zugeordnet ist und daß ein Abschnitt nicht über zwei Kapitel hinaus reicht. Bei einer solchen Organisation steht ein Block (hier: Abschnitt) vollständig in einem anderen Block (hier: Kapitel). Man bezeichnet eine solche Struktur kurz als *Blockstruktur*.

Schachtelung von HTML-Konstrukten

HTML

Im Hinblick auf die Struktur werden jetzt HTML-Konstrukte betrachtet. Ein HTML-Konstrukt ist definiert:

- durch eine (Anfangs-)Marke, notiert als „<bezeichner>“ und gegebenenfalls
- durch eine Endemarke, notiert als „</bezeichner>“.

Einige Konstrukte haben keine Endemarke oder diese Marke kann entfallen. Zusätzlich zum Bezeichner können Marken Attribute (Argumente) haben, denen über ein Gleichheitszeichen ein Wert zugewiesen werden kann. Der Wert ist in doppelte Hochkommata einzuschließen.[17] Bei der Angabe eines Wertes wird Groß/Klein-Schreibung unterschieden. Die Syntax für ein Konstrukt in HTML läßt sich in XML wie folgt notieren.

```
<!ELEMENT HTMLKONSTRUKT
  (ANFANGSMARKE, (TEXT? |
    (VORTEXT?,  HTMLKONSTRUKT, NACHTEXT?)*),
    ENDMARKE?)>
<!ELEMENT ANFANGSMARKE (BEZEICHNER, ATTRIBUT*)>
<!ELEMENT ATTRIBUT (ATTRIBUTNAME, GLEICH, VALUE)>
```

[17] Viele *Browser* benötigen jedoch die doppelten Hochkommata nicht (mehr).

Diese Blockstruktur ist rekursiv notiert, da Konstrukte geschachtelt werden können. Ein Konstrukt kann eine Sequenz von weiteren HTML-Konstrukten einschließen. So beinhaltet das Konstrukt `<HTML lang="de">` die Konstrukte `<HEAD>` und `<BODY>`. Das Attribut `lang` gibt hier an, daß das Dokument in Deutsch abgefaßt ist.[18] Das `<HEAD>`-Konstrukt hat üblicherweise das `<TITLE>`-Konstrukt zum Ausgeben eines Textes in der Überschriftszeile des Browsers. Ein einfaches Beispiel für die Schachtelung von HTML-Konstrukten zeigt die Datei `Hello.html`

`Hello.html`

```
 1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
 2    "http://www.w3c.org/TR/REC-html40/strict.dtd">
 3  <!-- Bonin 27-Sep-1999 -->
 4  <HTML lang="de">
 5    <HEAD>
 6      <TITLE>Simples HTML-Beispiel</TITLE>
 7      <LINK HREF="/myStyle.css"
 8        REL="stylesheet" TYPE="text/css">
 9    </HEAD>
10  <BODY>
11    <H1>Hello World!</H1>
12    <P> Gru&szlig; von <EM>as.fh-lueneburg.de</EM></P>
13  </BODY>
14  <!-- Quelle: 193.174.33.106:/u/bonin/mywww/ewi/widata/Hello.html --
15  </HTML>
16
```

Das HTML-Dokument beginnt mit einer Angabe des Dokumententypes in folgender Form:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
  "http://www.w3c.org/TR/Rec-html40/strict.dtd">
```

Hier kann nur stichwortartig diese zugegebenermaßen etwas kryptisch aussehende Typbeschreibung erläutert. Ein Identifier ist eine eindeutige Bezeichnung. Wird ihm das Schlüsselwort `PUBLIC` vorangestellt, dann ist es ein öffentlicher Identifier dessen Name für viele Plattformen aussagekräftig ist (beziehungsweise sein sollte). Der Identifier setzt sich aus einzelnen Bereichen zusammen, die durch doppelte Schrägstriche „//“

PUBLIC

[18] Das Attribut `lang` beschreibt das HTML-Dokument im Sinne von Meta-Daten. Näheres zu Meta-Daten ↪Abschnitt 2.4 Seite 105.

//

voneinander getrennt sind. Die ersten beiden Bereiche bezeichnen den Eigentümer (*owner identifier*). Steht am Anfang ein Bindestrich, dann handelt es sich um einen Identifier, der nicht formal registriert ist. In unserem Fall ist der Eigner das *WWW Consortium*[19]. W3C ist keine „Normungsinstanz“ wie es zum Beispiel die *International Standardization Organisation* (ISO) ist. Der folgende Bereich ist der sogenannte Text-Identifier. Sein erstes Wort verweist auf die Textklasse, in unserem Fall ist es `DTD`, weil es sich um eine *Document Type Definition* handelt.[20] Danach folgt die Textbeschreibung, in unserem Fall `HTML 4.0` für *Hypertext Markup Language* Version 4.0. Die Textbeschreibung wird gefolgt von der optionalen Angabe der Textsprache, in unserem Fall `EN` für Englisch.

W3C

HTML 4.0

Aus dem öffentlichen Identifier `"-//W3C//DTD HTML 4.0//EN"` wird versucht einen *Uniform Resource Identifier* (URI) zu konstruieren. Wenn dies nicht gelingt, dann wird der danach explizit angebene URI genutzt, in unserem Fall also:
`"http://www.w3c.org/TR/Rec-html40/strict.dtd"`.
In der Datei `strict.dtd` steht die eigentliche Spezifikation. Diese heißt `strict`, weil sie keine Elemente und Attribute zur Präsentation (Layout) enthält. Diese Aufgabe wird vollständig einem *Style Sheet* übertragen.[21] Tabelle A.12 auf Seite 268 zeigt einen Auszug der Datei `strict.dtd`.

URI

2.3.2 *Well-formed* & valide Daten

Ein HTML-Dokument ist *well-formed*, wenn es die obigen Konstruktionsregeln einhält. Dabei kann es durchaus vorkommen, daß vor der Kapitelüberschrift (`<H1>`-Konstrukt) eine Abschnittsüberschrift (`<H2>`-Konstrukt) steht. Eine solche Reihenfolge wäre kein Verstoß gegen die Schachtelungsvorschrift.

[19] W3C wird primär getragen von dem amerikanischen *Massachusetts Institute of Technology* (MIT ↪`http://www.mit.edu/` , Zugriff 17-Dec-1999), dem französischen *Institut National de Recherche en Informatique et en Automatique* (INRIA ↪`http://www.inria.fr/`, Zugriff 17-Dec-1999) und der japanischen *Keio University* (`http://www.keio.ac.jp/` Zugriff 17-Dec-1999). Ein W3C-Standard durchläuft die Verbindlichkeitsstufen: *Working Draft*, *Proposed Recommendation* und *Recommendation*.

[20] Eine andere Textklasse als `DTD` wäre zum Beispiel die Klasse `ENTITIES`.

[21] Wenn trotz alledem Elemente und Attribute zur Präsentation benötigt werden, dann benutzt man die *Transitional DTD*.

Bei einem XML-Dokument wird die Reihenfolge präzise in der *Document Type Definition* (DTD) festgelegt; zum Beispiel in folgender Art:

```
<!ELEMENT KAPITEL (H1, INHALT)>
<!ELEMENT INHALT (ABSCHNITT+)>
<!ELEMENT ABSCHNITT (H2, TEXT)>
```

Mit dieser DTD ist ausgeschlossen, daß ein `<H2>`-Konstrukt vor „seinem“ `<H1>`-Konstrukt stehen darf. Es lassen sich daher zwei Korrektheitsgrade unterscheiden:

well-formed: Einhaltung der korrekten Schachtelung
Sie ist ohne Rückgriff auf die DTD prüfbar.

valide: Erfüllung der zugehörenden DTD.
Die Validierung setzt die *well-formed*-Korrektheit voraus.

2.3.3 Bücherliste als XML-Beispiel

Als Beispiel einer Validierung dient eine in XML codierte Bücherliste. Ihre Daten enthält die Datei `booklist.xml` (↪Seite 100; aufbereitet ↪Bild 2.1 auf Seite 103). Ihre *Document Type Definition* (DTD) steht in der Datei `booklist.dtd` (↪Seite 96) und ihre Präsentationsaufbereitung in der Datei `booklist.xsl` (↪Seite 102). Hier interessiert uns, ob die DTD erfüllt wird. Da XML eine Anwendung[22] von SGML (*Standard Generalized Markup Language*) ist, läßt sich dazu ein SGML-Parser einsetzen. Ein Parser ist ein Programm zum Validieren und Zerlegen einer Datenstruktur. WI>Data nutzt den SGML-Parser `nsgmls` in der Version SP 1.3 von James Clark.[23] Diesen SGML-Parser wird in einem MS-DOS-Fenster wie folgt aufgerufen:

SGML

Parser

```
nsgmls -w xml -f errorbkl.txt booklist.xml
  > booklist.txt
```

Der Schalter `-w` sorgt dafür, daß vor Konstrukten gewarnt wird, die nicht XML-konform sind. Mit dem Schalter `-f` wird festgelegt, daß die die

[22] XML ≡ *„application profil or restriced form of SGML“*

[23] Homepage von James Clark: `http://www.jclark.com` Zugriff 29-Sep-1999

Warn- und Fehlermeldungen des Parsers in die Datei `errorbkl.txt` (↪Seite 101) geschrieben werden. Das Symbol `>` sorgt dafür, daß das Zerlegungsergebnis des Parsers in die Datei `booklist.txt` (↪Seite 101) umgeleitet wird und nicht auf dem Bildschirm (Standardausgabe) geschrieben wird. Mit diesem Aufruf übernimmt der SGML-Parser `nsgmls` zwei Aufgaben:

Validieren Stimmt die Datei `booklist.xml` mit der Definition von `booklist.dtd` überein?
Das Prüfergebnis wird in die Datei `errorbkl.txt` (↪Seite 101) geschrieben.

Zerlegung Erzeugen einer Darstellung der Elemente der Datei `booklist.xml` im *Element Structure Information Set*-Format (ESIS).
Das Ergebnis wird in die ESIS-Datei `booklist.txt` (↪Seite 101) geschrieben.

Die Validierung zeigt, daß die Struktur der Bücherliste mit ihrer Definition übereinstimmt. Der Parser erkennt die Verletzung der Eindeutigkeit des `ID`-Attributes (— zwei Bücher mit `CITE="Bo1991a"`). Nicht beantworten kann der Parser die inhaltliche Richtigkeit der Informationen; beispielsweise die Frage, ob das Buch mit der Identität `CITE="Bo1993a"` tatsächlich im Verlag `"North-Holland"` erschienen ist. Die mit dem Parser überprüfbare Korrektheit umfaßt daher nicht diese „Richtigkeit“ der Daten.

`booklist.dtd`

```
 1  <?xml version="1.0" encoding="ISO-8859-1"?>
 2  <!-- Bonin    21-Jul-1999                              -->
 3  <!--   update 30-Sep-1999,...,04-Oct-1999              -->
 4  <!ELEMENT BOOKLIST (BOOK+)>
 5  <!ELEMENT BOOK ((ISBN | ISSN), AUTHOR*, EDITOR*,
 6                  TITLE?, PUBLISHER+, RELEASED?)>
 7    <!ATTLIST BOOK CITE ID #REQUIRED>
 8  <!ELEMENT ISBN      (#PCDATA)>
 9  <!ELEMENT ISSN      (#PCDATA)>
10  <!ELEMENT AUTHOR    (#PCDATA)>
11    <!ATTLIST AUTHOR WRITING CDATA #FIXED "SELF">
12  <!ELEMENT EDITOR    (#PCDATA)>
13  <!ELEMENT TITLE     (#PCDATA)>
```

```
14  <!ELEMENT PUBLISHER  ANY>
15  <!ELEMENT RELEASED  (#PCDATA)>
16
17  <!-- Kurzschreibweise                                    -->
18  <!ENTITY Bonin "Bonin, Hinrich E.G.">
19
20  <!-- EIN-/AUS-Schalten mit "INCLUDE" und "IGNORE"   -->
21  <!ENTITY % BELEGTEZEICHEN "INCLUDE">
22  <![%BELEGTEZEICHEN; [
23  <!ENTITY lt   "&#38;#60;">
24  <!ENTITY gt   "&#62;">
25  <!ENTITY amp  "&#38;#38;">
26  <!ENTITY apos "&#39;">
27  <!ENTITY quot "&#34;">
28  ]]>
29  <!-- Quelle: /u/bonin/mywww/ewi/widata/booklist.dtd -->
30
```

In dieser DTD sind zwei Attributlisten definiert (↪Zeile 7 und Zeile 11). Eine Attributliste hat folgende Struktur:

`<!ATTLIST` element-name attribut-name attribut-daten angabe-notwendigkeit

. . . gegebenenfalls weitere attribute . . . `>`

Die zulässigen Angaben in XML Version 1.0 für attribut-daten beschreibt Tabelle 2.11 Seite 98. In Tabelle 2.12 Seite 98 sind die zulässigen XML-Angaben für die angabe-notwendigkeit beschrieben. Das Attribut `CITE` des Elementes `BOOK` muß daher für jedes Buch in der Datei `booklist.xml` einen unterschiedlichen Wert haben. `CITE` dient zur eindeutigen Identifizierung jedes beschriebenen Buches und hat daher die Angabe `ID`. Das Attribut `WRITING` des Elementes `AUTHOR` muß stets mit dem Wert `"SELF"` angegeben sein. Eine andere Wertangabe für ein Buch ist eine Verletzung dieser DTD.

ID

In Zeile 14 hat das Element `PUBLISHER` als Inhaltsangabe `ANY`. Die `ANY`-Angabe ermöglicht einen Elementinhalt, der aus Text, aus weiteren Elementen und aus einer Kombination davon bestehen kann. Mit einer `ANY`-Angabe kann praktisch nur noch die *well-formed*-Korrektheit überprüft werden.

ANY

In Zeile 18 wird das `ENTITY`-Konstrukt verwendet. Ein *Parsed General Entity* wird eingesetzt, um einen Text in das Dokument einzufügen. Es wird wie folgt in der DTD deklariert:

ENTITY

`<!ENTITY` entityName `"` entityInhalt `">`

Attribut-typ	Bedeutung
ID	Ein Attribut einer Attributliste kann den Typ ID haben. Der ID-Wert muß eindeutig im gesamten Dokument sein. Der ID-Wert muß mit einem Buchstaben beginnen und kann Ziffern, Buchstaben, Bindestriche, Unterstriche und Punkte enthalten.
IDREF	Der IDREF-Wert muß mit einem ID-Wert im Dokument übereinstimmen.
IDREFS	Der Attributwert muß mindestens einen Wert enthalten, der mit einem ID-Wert im Dokument übereinstimmen.
($wert_1$ | $wert_2$ | ...)	Eine Aufzählung der gültigen Werte für das Attribut.
NOTATION	Der Attributwert ist eine Liste von gültigen NOTATION-Identifiers.
CDATA	Jede XML-konforme Zeichenkette kann Attributwert sein.
ENTITY	Der Attributwert muß mit einer ENTITY-Deklaration in der DTD übereinstimmen.
ENTITIES	Wie ENTITY, jedoch können mehrere Bezeichner getrennt durch Leerzeichen angegeben werden.
NMTOKEN	Der Attributwert besteht aus Zeichen, die zum Bilden des Bezeichners einer Marke genutzt werden können.
NMTOKENS	Wie bei NMTOKEN; allerdings werden mehrere Werte getrennt durch Leerzeichen akzeptiert.

Legende: Näheres dazu zum Beispiel ↪[La+99]

Tabelle 2.11: Attributbeschreibung in XML Version 1.0

Attribut-wertangabe	Bedeutung
#IMPLIED	Jede Instanz des Elementes kann dieses Attribut haben.
#REQUIRED	Jede Instanz des Elementes muß dieses Attribut mit einem Wert haben.
#FIXED "wert"	Jede Instanz des Elementes muß dieses Attribut genau mit diesem Wert haben.
"ersatzWert"	Jede Instanz des Elementes erhält den Ersatzwert, wenn das Attribut nicht angegeben ist.

Legende: Näheres dazu zum Beispiel ↪[La+99]

Tabelle 2.12: Angabenotwendigkeit eines Attributes in XML Version 1.0

Im Dokument wird der entityInhalt an die Stelle eingefügt, an der der Name der ENTITY eingeschloßen in ein Ampersandzeichen „&“ und ein Semikolon „;“ steht, also folgendermaßen angegeben ist:
&entityName;

In Zeile 21 wird ebenfalls das ENTITY-Konstrukt verwendet. Hier dient als *Parameter Entity*. Ein *Parameter Entity* wirkt nur innerhalb der DTD und nicht auf das XML-Dokument. Auch es wirkt als Kurzschreibweise. Seine Syntax enthält das Prozentzeichen „%“. Zur Deklaration ist zu notieren:
<!ENTITY % entityName "entityInhalt">

Genutzt wird es mit folgender Notation:

%entityName;

Mit Hilfe der IGNORE- und INCLUDE-Konstrukte läßt sich eine DTD fallbezogen ausführen, ohne vorher mit einem Editor Bereiche in der DTD zu löschen oder einzufügen. Der mit IGNORE gekennzeichnete Bereich wird bei der Abarbeitung (Validierung) nicht berücksichtigt. Der mit INCLUDE gekennzeichnet Bereich wird ausgeführt. Die Bereiche werden durch eckige Klammern „[...]“ abgegrenzt; beispielsweise wie folgt:

IGNORE

INCLUDE

```
<![INCLUDE [
  <!ENTITY FHNON
    "University of Applied Sciences">
]]>
<![IGNORE [
  <!ENTITY FHNON
    "Fachhochschule Nordostniedersachsen">
]]>
```

In Zeile 22 werden mit dem Wert des Parameters %BELEGTZEICHEN die Deklarationen für das Kleinerzeichen „<“, Größerzeichen „>“, Ampersandzeichen „&“, Apostoph „'“ und Quotezeichen „"“ eingeschaltet. Diese Deklarationen für die Sonderzeichen sind häufig standardmäßig schon eingebaut.

booklist.xml

Einige Erläuterungen zum Zusammenwirken von booklist.xml und booklist.xsl ↪Seite 103.

```
 1  <?xml version="1.0" encoding="ISO-8859-1"?>
 2  <!-- Bonin 23-Jul-1999                  -->
 3  <!--   update 27-Sep-1999               -->
 4  <!DOCTYPE BOOKLIST SYSTEM "booklist.dtd">
 5  <?xml-stylesheet type="text/xsl" href="booklist.xsl"?>
 6  <BOOKLIST>
 7    <BOOK CITE="Bo1988a">
 8      <ISBN>3-8226-3187-6</ISBN>
 9      <AUTHOR WRITING="SELF">&Bonin;</AUTHOR>
10      <TITLE>Die Planung komplexer Vorhaben der Verwaltungsautomation</TITLE>
11      <PUBLISHER>R.v.Decker &amp; C.F.Müller</PUBLISHER>
12      <RELEASED>1988</RELEASED>
13    </BOOK>
14    <BOOK CITE="Bo1990a">
15      <ISBN>3-8226-0190-X</ISBN>
16      <EDITOR>&Bonin;</EDITOR>
17      <TITLE>Entmythologisierung von Expertensystemen</TITLE>
18      <PUBLISHER>R.v.Decker &amp; C.F.Müller</PUBLISHER>
19      <RELEASED>1990</RELEASED>
20    </BOOK>
21    <BOOK CITE="Bo1991a">
22      <ISBN>3-11-011786-X</ISBN>
23      <AUTHOR WRITING="???">&Bonin;</AUTHOR>
24      <TITLE>Software-Konstruktion mit LISP</TITLE>
25      <PUBLISHER>Walter de Gruyter</PUBLISHER>
26      <RELEASED>1991</RELEASED>
27    </BOOK>
28    <BOOK CITE="Bo1993a">
29      <ISSN>0926-5473</ISSN>
30      <EDITOR>&Bonin;</EDITOR>
31      <TITLE>Systems  Engineering in Public Administration</TITLE>
32      <PUBLISHER>North-Holland</PUBLISHER>
33      <RELEASED>1993</RELEASED>
34    </BOOK>
35    <BOOK CITE="Bo1991a">
36      <ISBN>3-446-18500-3</ISBN>
37      <AUTHOR>&Bonin;</AUTHOR>
38      <TITLE>&lt;HTML&gt;-Ratgeber</TITLE>
39      <PUBLISHER>Car Hanser Verlag</PUBLISHER>
40      <RELEASED>1996</RELEASED>
41    </BOOK>
42  </BOOKLIST>
```

```
43  <!-- Quelle: /u/bonin/mywww/ewi/widata/booklist.xml -->
44
```

errorbkl.txt

```
 1  nsgmls:booklist.dtd:4:26:E: omitted tag minimization parameter
 2    can be omitted only if "OMITTAG NO" is specified
 3    on the SGML declaration
 4  ...
 5  nsgmls:booklist.xml:23:24:E: value of fixed attribute
 6    "WRITING" not equal to default
 7  nsgmls:booklist.xml:35:14:E: ID "BO1991A" already defined
 8  nsgmls:booklist.xml:21:14: ID "BO1991A" first defined here
 9
```

booklist.txt

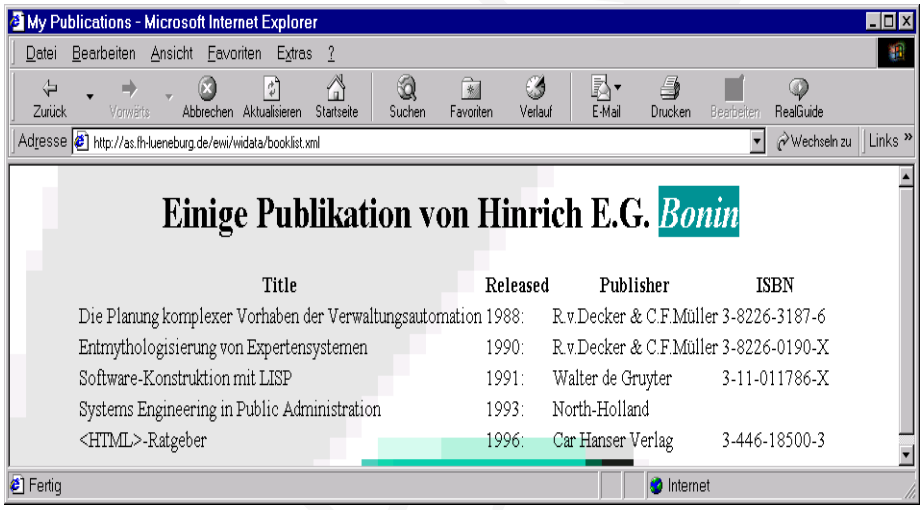
```
 1  ?xml version="1.0" encoding="ISO-8859-1"?
 2  ?xml version="1.0" encoding="ISO-8859-1"?
 3  ?xml-stylesheet type="text/xsl" href="booklist.xsl"?
 4  (BOOKLIST
 5  ACITE TOKEN BO1988A
 6  (BOOK
 7  (ISBN
 8  -3-8226-3187-6
 9  )ISBN
10  (AUTHOR
11  -Bonin, Hinrich E.G.
12  )AUTHOR
13  (TITLE
14  -Die Planung komplexer Vorhaben der Verwaltungsautomation
15  )TITLE
16  (PUBLISHER
17  -R.v.Decker & C.F.Müller
18  )PUBLISHER
19  (RELEASED
20  -1988
21  )RELEASED
22  )BOOK
23  ACITE TOKEN BO1990A
24  (BOOK
25  .
26  .
27  .
```

```
28  )BOOKLIST
29
```

booklist.xsl

Einige Erläuterungen zum Zusammenwirken von booklist.xml und booklist.xsl ↪Seite 103.

```
 1  <?xml version="1.0" encoding="ISO-8859-1"?>
 2  <!-- Bonin    01-Oct-1999 ... 13-Dec-2000          -->
 3  <xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl" version="1.0">
 4    <xsl:template match="/">
 5      <HTML>
 6        <HEAD>
 7          <TITLE>My Publications</TITLE>
 8          <STYLE TYPE="text/css">
 9            @import
10              url("http://as.fh-lueneburg.de/myStyle.css");
11            BODY {
12              background:
13                url(http://as.fh-lueneburg.de/logo.gif);
14              text-align: center;
15            }
16          </STYLE>
17        </HEAD>
18        <BODY>
19          <H1>Einige Publikationen von Hinrich E.G. <EM>Bonin</EM></H1>
20           <TABLE>
21             <THEAD>
22               <TR>
23                 <TH>Title</TH>
24                 <TH>Released</TH>
25                 <TH>Publisher</TH>
26                 <TH>ISBN</TH>
27               </TR>
28             </THEAD>
29             <TBODY>
30               <xsl:apply-templates select="//BOOK" />
31             </TBODY>
32           </TABLE>
33         </BODY>
34       </HTML>
35     </xsl:template>
36
37     <xsl:template match="BOOK">
38       <TR>
39        <xsl:apply-templates select="TITLE" />
40        <xsl:apply-templates select="RELEASED" />
```



Legende:
↪Quellcodelisting 86 auf Seite 100

Abbildung 2.1: XML-Beispiel: *Bücherliste*

```
41        <xsl:apply-templates select="PUBLISHER" />
42        <xsl:apply-templates select="ISBN" />
43      </TR>
44    </xsl:template>
45
46    <xsl:template match="TITLE | RELEASED | PUBLISHER | ISBN">
47      <TD>
48        <xsl:value-of />
49      </TD>
50    </xsl:template>
51
52  </xsl:stylesheet>
53  <!-- Quelle: /u/bonin/mywww/ewi/widata/booklist.xsl -->
54
```

Zusammenwirken: DTD⇔XML⇔XSL⇔HTML⇔CSS

Ein Vorteil von XML-Dokumenten gegenüber den ersten HTML-Dokumenten[24] ist ihre Freiheit von Angaben über die Darstellung des Doku-

[24] Notiert in HTML 1.0 bis 3.2 — in HTML 4.0 ist die Trennung von Präsentation des Dokumentes und Dokumentinhalt gegeben.

mentes. Während mit der DTD eines XML-Dokumentes die Validierung des Dokumentes möglich ist, fehlt die Präsentationsspezifikation. Diese Aufgabe wird in ein sogenanntes *Style Sheet* verlagert. Dabei stehen zwei *Style-Sheet*-Konzepte zur Wahl:

CSS *Cascading Style Sheets*[25]
CSS ist ein relativ einfacher Mechanismus um Fonts, Farben, Platz und Bilder einem Dokument zuzuordnen. Dabei sind Layoutprioritäten im Verhältnis von Autoren und Leser spezifizierbar. CSS beinflußt das Dokument inhaltlich nicht. Zum Beispiel kann keine Änderung der Reihenfolge von Konstrukten erfolgen.

XSL *Extensible Stylesheet Language*[26]
Die XSL-Aufbereitung geschieht in zwei Schritten.

1. In der *Tree-Transformation*-Phase wird ein neuer Ergebnisbaum von dem Ausgangsdokument erzeugt. XSL transformiert also das XML-Dokument in ein anderes und kann dabei beispielsweise die Reihenfolge der Konstrukte ändern. Dazu sind in XSL Muster definiert, die auf einzelne Konstrukte (genauer Knoten) des Ausgangsdokumentes passen. Die Muster sind mit Regeln verknüpft, welche die Transformation in den Ergebnisbaum vollziehen.
2. In der *Formatting*-Phase wird der Ergebnisbaum interpretiert, um die Präsentation für Bildschirm, Papier, Sprache oder ein anderes Medium zu erzeugen.

Um ein XML-Dukumente zu präsentieren, können beide Konzepte gemeinsam genutzt werden. Mit XSL-Hilfe wird ein HTML-Dokument erzeugt, das in seinem `<STYLE>`-Konstrukt CSS-Angaben enthält. Diese Mix-Konstruktion weist das obige Beispiel `booklist.xsl` auf. Erzeugt wird aus dem XML-Dokument `booklist.xml` ein HTML-Dokument. Beim „Lesen‘ des `root`-Knotens des XML-Dokumentes —

[25] Zum Einstieg in die CSS-Literatur ↪`http://www.w3.org/Style/CSS/`, Zugriff 21-Dec-1999.

[26] Zum Einstieg in die XSL-Literatur ↪`http://www.w3.org/Style/XSL/`, Zugriff 21-Dec-1999.

präziser formuliert: wenn beim Mustervergleich (*matching*) der *root*-Knoten vorliegt —, werden die HTML-Konstrukte generiert. Das *Root-Node-Matching* wird wie folgt notiert:

template

```
<xsl:template match="/">
...
... Angaben zum HTML-Dokument mit CSS
...
</xsl:template>
```

Mit dem <xsl:apply-templates>-Konstrukt wird die Knotenverarbeitung angestoßen. Liegt zum Beispiel der Knoten, der durch das XML-Konstrukt <ISBN>3-8226-3187-6</ISBN> erzeugt wurde, vor, dann wird mit der Angabe:

apply-template

```
<xsl:apply-templates select="ISBN" />
```

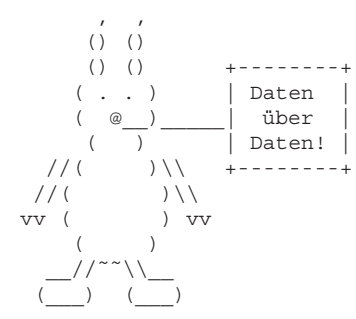
das folgende Muster appliziert:

```
<xsl:template match="ISBN">
  <TD>
    <xsl:value-of />
  </TD>
</xsl:template>
```

Dabei wird an die Stelle von <xsl:value-of /> der jeweilige Knotenwert gesetzt, also hier die in seinem <ISBN>-Konstrukt angegebene Nummer 3-8226-3187-6. Wie das HTML-Tabellenelement für diesen ISBN-Wert auf dem Bildschirm dargestellt wird, ergibt sich aus der Hierarchie der Style-Angaben. Das <STYLE>-Konstrukt selbst weist keine Spezifikation für das <TD>-Konstrukt auf. In der CSS-Datei myStyle.css, die über das @import-Konstrukt geladen wird, sei ebenfalls keine Angabe zum Tabellenelement spezifiziert. Dann kommt die Style-Spezifikation des Browser, also dessen Default-Einstellung, zur Anwendung.

2.4 Daten über Daten: Meta-Daten

Meta-Daten sind Wegweiser zu den Daten



Meta-Daten sind Daten über Daten. Zum Beispiel bildet ein Bibliothekskatalog Meta-Daten ab, weil der Katalog Daten über die Bücher der Bibliothek, die eigentlichen Daten (≡ *Resources*), enthält. Meta-Daten sollen Daten beschreiben und zwar nicht nur für Menschen, sondern auch für eine maschinelle Auswertung; also auch für Programme. Meta-Daten bedürfen daher einer präzisen Spezifikation.

Ein Programm kann die Meta-Daten nur interpretieren („verstehen“), wenn eindeutig definiert ist, welche Bedeutung (Semantik) jedes einzelne Meta-Datum hat.

2.4.1 HMTL: <META>

In HTML Version 4.0 kann ein Dokument im Sinne von Meta-Daten durch ein eigenes Konstrukt, das <META>-Konstrukt, und durch zusätzliche Angaben (Attribute) in den einzelnen Konstrukten beschrieben werden. Ein solches „Meta-Daten-Attribut“ ist lang. Es spezifiziert die Sprache für seinen Konstruktbereich. Die Tabelle A.13 Seite 269 zeigt einige Werte für das lang-Attribut. Das <META>-Konstrukt in HTML beschreibt seine Resource, das heißt dasselbe Dokument, mit Hilfe von Attributen[27]. Typisch sind die beiden Attribute name und content. Ein <META>-Konstrukt wird in dem <HEAD>-Konstrukt angegeben, also zum Beispiel folgendermaßen:[28]

name

content

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
  "http://www.w3c.org/TR/REC-html40/strict.dtd">
```

[27] Das HTML-Konstrukt <META> ist aus XML-Sicht ein leeres Element:
<!ELEMENT META EMPTY>

[28] Hinweis: Wenn eine Beschreibung formuliert als <META>-Konstrukt eine URL-Angabe enthält, dann kommt auch das <LINK>-Konstrukt in Betracht; zum Beispiel statt:
<META name="foo" content="http://www.myServer.de/baz.txt"
steht dann:
<LINK rel="foo" type="text/plain"
href="http://www.myServer.de/baz.txt">

```
<!-- Bonin 06-Dec-1999                                    -->
<HTML lang="de">
<HEAD>
<BASE href="http://as.fh-lueneburg.de/">
<META name="description"
  content="Mustermann's Hompepage">
<META name="keywords"
  content="Wirtschaftsinformatik,Web-Technologie">
...
</HEAD>
<BODY>
...
</BODY>
</HTML>
```

Anstatt des Attributes `name` kann das Attribut `http-equiv` gesetzt werden. Manche Web-Server benutzen dann diese Attributangabe zur Generierung von entsprechenden Kopfdaten für das *Hypertext Transfer Protocol* (HTTP[29]). Zwei `http-equiv`-Beispiele zeigen die folgenden Zeilen:

http-equiv

```
<META http-equiv="Last-Modified"
      content="06-Dec-1999 13:00:00 GMT">
<META http-equiv="Expires"
      content="31-Dec-2000 00:00:00 GMT">
```

Herauszufinden wo die jeweilige *Resource*, zum Beispiel ein „Buch“, überall in einem Katalog zu plazieren ist, stellt eine nicht triviale Aufgabe dar. Im Zweifel werden daher häufig ein paar Attributangaben mehr als inhaltlich gerechtfertig wären formuliert. In der Praxis hat sich das `<META>`-Konstrukt zunehmend zu einem kommerziellen Kampfkonstrukt für die besten Eintragungen in Suchmaschinen entwickelt.

Robot-Steuerung

Robots

Mit dem `<META>`-Konstrukt können Programme, die Daten für Suchmaschinen ermitteln, sogenannte *Robots*, sehr einfach vom Autor des

[29] Näheres zu HTTP-Kopfdaten ↪Tabelle 4.4 Seite 203.

Dokumentes gesteuert werden.[30] Mit dem Wert `index` oder `noindex` läßt sich die Indizierung des betroffenen Web-Dokumentes erlauben oder unterbinden — natürlich nur, falls sich das Robot-Programm an diese Regelung hält. Der Wert `follow` oder `nofollow` dient zur Steuerung der Verfolgung von Verweisen (*Links*) in dem betroffenen Dokument. Für eine Web-Publikation, die in einer Suchmaschine vollständig referenziert werden soll, ist dann im `<HEAD>`-Konstrukt zu formulieren:

```
<META name="robots"
      content="index,follow">
```

2.4.2 Resource Description Framework

Eine entscheidende Schwachstelle dieses einfachen `name–content`-Systems ist die Mißverständlichkeit der gewählten Werte. Ein Verbesserungsansatz ist das *Resource Description Framework* (RDF). Es ist eine Grundlage für die präzise Beschreibung von Meta-Daten. RDF ermöglicht übergreifend über fachgebietsbezogene Anwendungen „maschinell verstehbare“ Meta-Daten auszutauschen. RDF benutzt XML um die Meta-Daten abzubilden. RDF wird federführend vom *World Wide Web Consortium* (W3C[31]) koordiniert und zielt daher zunächst auf die *Resources* im Web. Dabei geht es beispielsweise um die Aufgaben:

W3C

Finden: Finden von Web-Quellen (*Resource Discovery*) — Verbesserung der Ergebnisse von Suchmaschinen,

Ordnen: Katalogisieren — Beschreibung des Inhaltes und der Beziehungen von Web-Publikationen, von einer Sammlung von Web-Dokumenten

[30] Eine Alternative stellt das *Robots Exclusion Protocol* dar. Dabei wird eine zusätzliche Datei mit dem Namen `robots.txt` auf dem Web-Server eingerichtet; und zwar so daß der Zugriff `http://myServer.de/robots.txt` für die *Robots* erfolgreich ist. Die Datei `robots.txt` hat beispielsweise folgenden Inhalt:

```
# robots.txt for http://as.fh-lueneburg.de/
User-agent: *
Disallow: /tmp/ # these will soon disappear
```

Näheres dazu zum Beispiel:
`↪http://info.webcrawler.com/mak/projects/robots/exclusion-admin.html`
Zugriff 07-Dec-1999.

[31] W3C-Homepage: `http://www.w3.org` Zugriff 26-Nov-1999; Näheres zu W3C ↪Fußnote Seite 94.

und/oder von einzelnen Web-Dokumenten untereinander,

Rechte: Beschreiben von Rechten — zum Beispiel das Copyright für eine *Resource*,

Präferenz: Beschreiben von persönlichen Präferenzen eines Benutzers und/oder eines Web-Servers.

RDF besteht aus den beiden Teilen:

Syntax *Resource Description Framework (RDF) Model and Syntax Specification*[32]

Schema *Resource Description Framework (RDF) Schema Specification*[33] Ein RDF-Schema hat eine Aufgabe wie eine DTD bei XML. Es spezifiziert daher Meta-Daten für Meta-Daten; also **Meta-Meta-Daten**.

RDF ist beeinflußt worden von unterschiedlichen Disziplinen und Aktivitäten. Der Haupteinfluß kam und kommt sicherlich von den „Web-Akteure“ (*Internet Community*): einerseits von der HTML-Standardisierung mit dem `<META>`-Konstrukt und andererseits von den Arbeiten an der *Platform for Internet Content Selection*[34] (PICS). Zwei andere Wurzeln stammen aus dem Bibliothekswesen und dem Druckwesen. Sie prägten durch ihre Erfahrungen mit SGML und davon abgeleitet mit XML. Aus dem Bereich Wissensrepräsentation kam das Frame-System-Konzept[35]. Aus der Softwaretechnik (*Software Engineering*) wurde die objekt-orientierte Modellierung und die objekt-orientierte Programmierung mit dem Klassen- und Vererbungskonzept übernommen.

PICS

Die RDF-Quellen spiegeln diese vielfältigen Wurzeln wieder. Mal wird von Objekt aus der objekt-orientierten Softwaresicht gesprochen mal von Objekt aus der Sicht der Linguistik mal von Dokument und mal

[32] Quelle ↪`http://www.w3.org/TR/REC-rdf-syntax` — W3C Recommendation 22 February 1999; Zugriff 23-Nov-1999.

[33] Quelle ↪`http://www.w3.org/TR/PR-rdf-schema` — W3C Proposed Recommendation 03 March 1999; Zugriff 23-Nov-1999.

[34] PICS entstand ursprünglich um Eltern und Lehrern zu helfen, den Internetzugriff der Kinder zu kontrollieren. PICS spezifiziert Labels (Meta-Daten), die mit einem Internet-Inhalt verknüpft sind.

[35] RDF spezifiziert allerdings keinen Mechanismus für ein maschinelles *Reasoning* auf der Basis der Wissensrepräsentation (*Knowledge Representation*).

von *Resource*. Festgelegt sind in der RDF-Terminologie die folgenden drei Begriffe („Objekttypen“):

Resource

Jedes Ding (*Thing*), das von einem RDF-Ausdruck beschrieben werden kann, ist eine Resource. Eine Resource kann daher das HTML-Dokument wie `http://as.fhnon.de/index.html` sein. Eine Resource kann auch ein einzelnes HTML-Konstrukt oder ein einzelnes XML-Konstrukt in diesem HTML-Dokument sein. Eine Resource kann aber auch eine ganze Web-Publikation, also eine Sammlung von Web-Dokumenten sein. Stets hat eine Resource einen Namen in Form eines *Universal Resource Identifier* (URI); gegebenenfalls ergänzt durch den Identifier eines Ankers.

URI

Property

Eine Eigenschaft (*Porperty*) ist ein spezifischer Aspekt, ein Charaktersitikum, ein Atrribut, oder eine Relation um eine Resource zu beschreiben. Jede Eigenschaft hat eine bestimmte Bedeutung. Sie definiert:

- ihre zulässigen Werte,
- die Typen der Resourcen, die sie beschreiben kann und
- die Beziehungen zu anderen Eigenschaften.

Statement

Eine Resource zusammen mit der Eigenschaft (Property) und ihrem Wert (Value) für diese Eigenschaft bilden ein RDF-Statement. In XML notiert:

```
<!ELEMENT STATEMENT (RESOURCE, PROPERTY, VALUE)>
```

Aus der Sprachperspektive setzt sich ein Statement aus den Sprachkomponenten Subjekt, Prädikat und Objekt zusammen. Mit einem Beispielsatz wie: „Emma Musterfrau hat das Web-Dokument `foo.html` kreiert.“ werden diese RDF-Begriffe deutlicht.

Resource	$\equiv$	Subjekt:	`http://myServer.de/foo.html`
Eigenschaft	$\equiv$	Prädikat:	Creator
Wert (String)	$\equiv$	Objekt:	¨Emma Musterfrau¨

Als Beispiel spezifiziert die XML-Datei `myRDF.xml` RDF-gemäß die folgende Aussage über das Dokument `content.html` auf dem Web-Server `as.fh-lueneburg.de`: „Hinrich Bonin hat das Dokument kreiert. Er hat die Email-Adresse `hinrich-bonin@fbw.fh-lueneburg.de`." Dabei wird die Email-Adresse über die *Resource* der hauptamtlichen Mitarbeiter der Hochschule referenziert. Diese Meta-Daten haben also eine Baumstruktur. Das Prädikat `Creator` wird selbst über eine RDF-Beschreibung spezifiziert.

`myRDF.xml`

```
 1  <?xml version="1.0"?>
 2  <!-- Bonin 08-Dec-1999   -->
 3  <rdf:RDF
 4    xmlns:rdf=
 5      "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 6    xmlns:rdfs=
 7      "http://www.w3.org/TR/1999/PR-rdf-schema-19990303#">
 8    <rdf:Description about=
 9      "http://as.fh-luenburg.de/content.html">
10      <rdfs:Creator>
11        <rdf:Description about=
12  "http://www.fh-lueneburg.de/fbw/Mitarbeiter/HauptamlichLehrende/bo
13          <rdfs:Name>Hinrich Bonin</rdfs:Name>
14          <rdfs:Email>
15            hinrich-bonin@fbw.fh-lueneburg.de
16          </rdfs:Email>
17        </rdf:Description>
18      </rdfs:Creator>
19    </rdf:Description>
20  </rdf:RDF>
21  <!-- End of Object /u/bonin/mywww/ewi/myRDF.xml -->
22
```

2.4.3 Dublin Core Metadata Initiative

Die *Dublin Core Metadaten Initiative*[36] (DC) stellt ein Modell zur Beschreibung des Inhaltes einer elekronischen *Resource* bereit. Dabei ist jedes DC-Element zur Inhaltsbeschreibung selbst über Attribute eindeutig spezifiziert. Jedes DC-Meta-Daten-Element (in der Version 1.1) ba- **DC**

[36] Homepage ↪`http://purl.org/dc` Zugriff 23-Nov-1999

siert auf 10 Attributen. Diese entsprechen dem Standard ISO/IEC 11179, der zur Beschreibung von Datenelementen weitere Attribute umfaßt. Dabei handelt es sich um die folgenden Angaben:

Beschreibung der DC-Meta-Meta-Daten

Name Der Bezeichner (*Label*), der mit dem Datenelement verknüpt.

Identifier Der eindeutige Identifier, der mit dem Datenelement verknüpft ist.

Version Die Version des Datenelementes.
DC-Wert: `1.1`

Registration Authority Die Regestrierungsorganisation des Datenelementes.
DC-Wert: `Dublin Core Metadata Initiative`

Language Die Sprache in der das Datenelement spezifiziert ist.
DC-Wert: `en` ↪Tabelle A.13 Seite 269

Definition Ein Statement, das eindeutig und unmißverständlich das Konzept und den grundlegenden Charakter („*essential nature*“) des Datenelementes abbildet.

Obligation Gibt an, ob das Datenelement stets vorkommen muß oder nur vorkommen kann.
DC-Wert: `Optional`

Datatype Gibt den Typ der Daten an, die den Datenelementwert abbilden.
DC-Wert: `Character String`

Maximum Occurrence Gibt die Menge der Wiederholbarkeit des Datenelementes an.
DC-Wert: `Unlimited`

Comment Eine Anmerkung zur Applikation des Datenelementes.

Beschreibung der DC-Meta-Daten

Die DC-Meta-Daten-Elemente sind mit den genannten Attributen präzise spezifiziert. Hier sind sie aus Platzgründen nur mit ihren Anfangsmarken genannt; zum Beispiel:

```
<dc:Title>
```

statt vollständig:
`<dc:Title>...</dc:Title>`
Auch ist hier auf Platzgründen ihre umfangreiche Spezifikation auf eine kurze textliche Erläuterung reduziert.

`<dc:Title>` Der Titel unter dem die *Resource* „bekannt“ ist (oder werden soll).

`<dc:Creator>` Der für den Inhalt Verantwortliche (natürliche oder juristische Person).

`<dc:Subject>` Das Thema der *Resource*; üblicherweise ein Satz zur Klassifikation oder Schlagwörter eines vorgegebenen Kataloges (Thesaurus).

`<dc:Description>` Eine Beschreibung, üblicherweise im Sinne einer Zusammenfassung (*Abstract*).

`<dc:Publisher>` Der für die Verfügbarmachung Verantwortliche (natürliche oder juristische Person).

`<dc:Contributor>` Der für Beiträge Verantwortliche (natürliche oder juristische Person).

`<dc:Date>` Ein Datum, das mit dem Lebenszyklus der *Resource* verknüpft ist. Üblicherweise wird das Entstehungsdatum oder das Verfügbarkeitsdatum in der Form `YYYY-MM-DD` gewählt.[37]

`<dc:Type>` Gibt eine Klassifikation im Sinne einer Angabe über eine allgemeine Kategorie, Funktion, Gattung, Art oder auch über den

[37]
- **Jahr:** `YYYY` (z. B.: 2000)
- **Jahr und Monat:** `YYYY-MM` (z. B.: 2000-04)
- **Vollständiges Datum:** `YYYY-MM-DD` (z. B.: 2000-04-01)
- **Datum plus Stunde und Minute:** `YYYY-MM-DDThh:mm`TZD (z. B.: 2000-04-01T13:30+01:00)
- **Datum plus Stunde, Minute und Sekunde:** `YYYY-MM-DDThh:mm:ss`TZD (z. B.: 2000-04-01T13:30:11+01:00)
- **Datum plus Stunde, Minute, Sekunde und Sekundenbruchteil:** `YYYY-MM-DDThh:mm:ss.s`TZD (z. B.: 2000-04-01T13:30:11.25+01:00)

Mit TZD ≡ Zeitzone (`+hh:mm or -hh:mm`). Näheres zur Datum- und Zeitangabe ↪`http://www.w3.org/TR/NOTE-datetime` Zugriff 10-Dec-1999.

Aggregationsgrad des Inhaltes. Näheres dazu ↪*Dublin Core Types*[38].

`<dc:Format>` Die physikalische oder digitale Abbildung der *Resource*; üblicherweise als MIME-Typ[39] angegeben.

`<dc:Identifier>` Ein eindeutiger Verweis zur *Resource* in dem gegebenen Kontext; üblicherweise eine Angabe als *Uniform Resource Identifier* (URI) beziehungsweise als *Uniform Resource Locator* (URL) oder als *Digital Object Identifier* (DOI) oder als *International Standard Book Number* (ISBN).

`<dc:Source>` Ein Verweis auf die Quelle (*Resource*) von der die aktuelle *Resource* abgeleitet ist.

`<dc:Language>` Die Sprache in der der Inhalt der *Resource* formuliert ist; üblicherweise wird sie in einer zwei Buchstabenabkürzung (↪Seite 269) gemäß ISO639:1988 angegeben.

`<dc:Relation>` Ein Verweis zu einer betroffen anderen *Resource*.

`<dc:Coverage>` Die „Reichweite“ des Inhaltes der *Resource*; beispielsweise eine Angabe gemäß dem *Getty Thesaurus of Geographic Names* (TGN)[40]

`<dc:Rights>` Gibt Informationen an über die Rechte, die mit der *Resource* verbunden sind; üblicherweise geht es um die Beschreibung von *Intellectual Property Rights* (IPR) und um *Copyrights*.

eBook

Viele Projekte nutzen diese DC-Meta-Daten als Beschreibungsbasis für ihre Resourcen. Ein Beispiel ist die *Open eBook Initiative*[41]. Sie spezifiziert die Struktur für die Publikation eines elektronischen Buches. Ein *Open eBook* hat folgende Struktur:

[38] *Dublin Core Types* ↪`http://purl.oclc.org/docs/core/documents/wd-typelist.htm` Zugriff 08-Dec-1999

[39] Näheres zum MIME-Typ ↪Abschnitt 4.3.8 Seite 220.

[40] Der *Getty Thesaurus of Geographic Names* (TGN) ist ein strukturiertes Wörterverzeichnis, daß primär aus dem Bereich Geschichte entwickelt wurde. Näheres zu TGN ↪`http://shiva.pub.getty.edu/tgn_browser/`

[41] Homepage: `http://www.openebook.org/` Zugriff 10-Dec-1999

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Bonin: 10-Dec-1999                    -->
<!DOCTYPE package
  PUBLIC "+//ISBN 0-9673008-1-9//DTD OEB 1.0 Package//EN"
  "http://openebook.org/dtds/oeb-1.0/oebpkg1.dtd">
<package>
  ...Meta-Daten...
  ...Manifest (alle Dateien der Publikation)...
  ...Lineare Lesereihenfolge ("Spine")...
  ...Spezielle Zusammenstellungen ("Tours")...
  ...Referenzen (z.B. Inhaltsverzeichnisse)...
</package>
```

Beispiel: DC-Meta-Daten im `<rdf:RDF>`-Konstrukt

Die folgende Datei `WIDATA.RDF` zeigt beispielhaft die Anwendung dieser DC-Beschreibung im `<rdf:RDF>`-Konstrukt (↪Seite 108) einer HTML-Datei.

`WIDATA.RDF`

```
 1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
 2    "http://www.w3c.org/TR/REC-html40/strict.dtd">
 3  <HTML>
 4  <!--  Bonin:  10-Dec-1999                    -->
 5  <HEAD>
 6  <rdf:RDF
 7      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 8      xmlns:dc="http://purl.org/metadata/dublin_core#">
 9      <rdf:Description about=
10        "http://as.fh-lueneburg.de/ewi/WIDATA.RDF">
11        <dc:Title>WI&gt;DATA</dc:Title>
12        <dc:Description>
13          WI&gt;DATA ist eine Einf&uuml;hrung in die
14          Wirtschaftsinformatik f&uuml;r Studierende und DV-Praktiker.
15          WI&gt;DATA erl&auml;utert die Fachdisziplin mit Hilfe
16          von HTML, XML, UML und Java.</dc:Description>
17        <dc:Publisher>
18          University of Applied Sciences
19          L&uuml;neburg</dc:Publisher>
20        <dc:Date>1999-11-23</dc:Date>
21        <dc:Creator>Bonin, Hinrich E. G.</dc:Creator>
```

```
22        <dc:Subject>
23          <rdf:Bag>
24            <rdf:li>Wirtschaftsinformatik</rdf:li>
25            <rdf:li>Web</rdf:li>
26            <rdf:li>Java</rdf:li>
27            <rdf:li>UML</rdf:li>
28            <rdf:li>XML</rdf:li>
29            <rdf:li>HTML</rdf:li>
30          </rdf:Bag>
31        </dc:Subject>
32        <dc:Type>Text.Manuscript</dc:Type>
33        <dc:Format>application/postscript</dc:Format>
34        <dc:Language>de</dc:Language>
35      </rdf:Description>
36    </rdf:RDF>
37  <LINK HREF="http://as.fh-lueneburg.de/myStyle.css"
38    REL="stylesheet" TYPE="text/css">
39  </HEAD>
40  <BODY>
41  <H1>Publikation <EM>WI&gt;DATA</EM></H1>
42  ...
43  </BODY>
44  <!-- Ende der Datei /u/bonin/mywww/ewi/widata/WIDATA.RDF -->
45  </HTML>
46
```

Beispiel: DC-Meta-Daten im `<META>`-Konstrukt

Die folgende Datei `WIDATA.html` zeigt beispielhaft die Anwendung dieser DC-Beschreibung im `<META>`-Konstrukt (↪Seite 106) einer HTML-Datei.

`WIDATA.html`

```
 1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
 2    "http://www.w3c.org/TR/REC-html40/strict.dtd">
 3  <HTML>
 4  <!--  Bonin:  08-Dec-1999                    -->
 5  <HEAD>
 6  <META NAME="DC.Title" CONTENT="WI&gt;DATA">
 7  <LINK REL=SCHEMA.dc
 8    HREF=
 9      "http://purl.org/metadata/dublin_core_elements#title">
10  <META NAME="DC.Title.Alternative"
11    CONTENT="Eine Einf&uuml;hrung in die
12    Wirtschaftsinformatik auf der Basis der Web-Technologie">
```

```
13  <LINK REL=SCHEMA.dc
14    HREF=
15      "http://purl.org/metadata/dublin_core_elements#title">
16  <META NAME="DC.Creator"
17    CONTENT="Bonin, Hinrich E. G.">
18  <LINK REL=SCHEMA.dc
19    HREF=
20     "http://purl.org/metadata/dublin_core_elements#creator">
21  <META NAME="DC.Creator.Address"
22    CONTENT="hinrich-bonin@fbw.fh-lueneburg.de">
23  <LINK REL=SCHEMA.dc
24    HREF=
25      "http://purl.org/metadata/dublin_core_elements#creator">
26  <META NAME="DC.Subject" CONTENT="Wirtschaftsinformatik">
27  <LINK REL=SCHEMA.dc
28    HREF=
29      "http://purl.org/metadata/dublin_core_elements#subject">
30  <META NAME="DC.Subject" CONTENT="Web">
31  <LINK REL=SCHEMA.dc
32    HREF=
33      "http://purl.org/metadata/dublin_core_elements#subject">
34  <META NAME="DC.Subject" CONTENT="Java">
35  <LINK REL=SCHEMA.dc
36    HREF=
37      "http://purl.org/metadata/dublin_core_elements#subject">
38  <META NAME="DC.Subject" CONTENT="UML">
39  <LINK REL=SCHEMA.dc
40    HREF=
41      "http://purl.org/metadata/dublin_core_elements#subject">
42  <META NAME="DC.Subject" CONTENT="XML">
43  <LINK REL=SCHEMA.dc
44    HREF=
45      "http://purl.org/metadata/dublin_core_elements#subject">
46  <META NAME="DC.Subject" CONTENT="HTML">
47  <LINK REL=SCHEMA.dc
48    HREF="http://purl.org/metadata/dublin_core_elements#subject">
49  <META NAME="DC.Description"
50    CONTENT="WI&gt;DATA ist eine
51    Einf&uuml;hrung in die Wirtschaftsinformatik
52    f&uuml;r Studierende und DV-Praktiker. WI&gt;DATA
53    erl&auml;utert die Fachdisziplin mit
54    Hilfe von HTML, XML, UML und Java.">
55  <LINK REL=SCHEMA.dc
56    HREF=
57      "http://purl.org/metadata/dublin_core_elements#description">
58  ...
```

```
59  <TITLE>WI&gt;DATA</TITLE>
60  <LINK HREF="http://as.fh-lueneburg.de/myStyle.css"
61    REL="stylesheet" TYPE="text/css">
62  </HEAD>
63  <BODY>
64  <H1>Publikation <EM>WI&gt;DATA</EM></H1>
65  ...
66  </BODY>
67  </HTML>
68
```

2.5 Datenmengenproblem: *Data Warehouse*

Bei der Abbildung eines real existierenden Geschäftsfeldes in Daten ist außer den vielfältigen Verküpfungen der einzelnen Daten und ihrer Meta-Daten stets die Menge ein kritischer Faktor. In großen Unternehmen verschärft sich dieses Problem, weil die betriebenen operativen Anwendungsysteme die Daten in unterschiedlichen Formaten und Definitionen speichern und verarbeiten. Es gibt daher in der Regel eine heterogene Daten- und Systemlandschaft. Trotz der großen Datenmengen und der unterschiedlichen Abbildungen in den einzelnen Sparten und Funktionsbereichen eines Unternehmens soll eine übergreifende Datenauswertung zeitgerecht ermöglicht werden. Ein Lösungsansatz dazu ist das sogenannte *Data Warehouse*[42]

heterogene Landschaft

Das *Data Warehouse* organisiert die große Menge von Daten und Meta-Daten mit dem Ziel diese im betrieblichen Alltag für Entscheidungen nutzen zu können. Dazu werden Daten aus den operativen Datenbeständen selektiert und aggregiert („destilliert“) und um Daten aus externen Quellen ergänzt. Ein *Data Warehouse* gibt seinen Benutzern eine dialog-orientierte Auskunft über die Inhalte, Formate und Auswertungsmöglichkeiten der entscheidungsrelevanten Daten. Es ist mehr als ein DBMS mit einer großen Datenbasis, die auch Meta-Daten enthält. Es umfaßt zusätzlich die notwendigen Werkzeuge (Tools), das heißt, die Anwendungssoftware, um die Daten verknüpfen, analysieren und präsentieren zu können. In XML notiert stellt sich das *Data Warehouse* wie folgt dar:

```
<!ELEMENT DATAWAREHOUSE (DATABASE, TOOL)>
<!ELEMENT DATABASE (DBMS+, DATA+, METADATA+)>
```

[42] Hinweis: Die deutsche Übersetzung „Daten-Lagerhaus“ ist unüblich ↪[Han96] S. 976.

Im Zusammenhang mit dem *Data-Warehouse*-Konzept steht der *Data Mart*. Ein *Data Mart* ist in der Regel gekennzeichnet durch ein einzelnes Produkt (Subjekt), eine einzelne Geschäftsfunktion und/oder durch eine einzelne Anwendung. Im Vergleich zielt das *Data Warehouse* auf eine „umfassendere“ Aufgabenstellung.

2.5.1 Datenwürfel: *On-line Analytical Processing*

Das *Data Warehouse* bildet den Rahmen für das *On-line Analytical Processing* (OLAP). E. F. Codd, der Konzepteur des *Relational Model for Large Shared Data Banks* [Codd70], prägte 1993 diesen Begriff.[43] Er spezifizierte für OLAP Regeln die mit Akronym FASMI (*Fast Analysis of Shared Multidimensional Information*) charakterisierbar sind. Mit OLAP soll ein Nutzer (Manager) schnell (*fast*) im Mehrbenutzerbetrieb (*shared*) komplexe Datenverknüpfungen (*multidimensional*) entscheidungsbezogen analysieren können. Mindestens bei großen Datenmengen sind die notwendigen Selektions-, Sortier- und Aggregationsverfahren zu aufwendig, um sie während einer dialogorientierte Analyse durchzuführen.

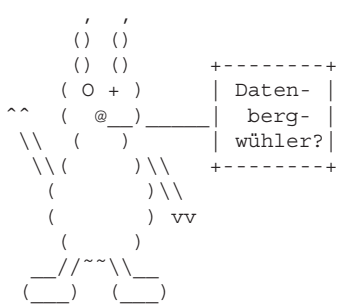
FASMI

Eine angemessene Dialogantwortzeit kann nur gewährleistet werden, wenn die Daten im *Data Warehouse* für die Analyse paßend vorverarbeitet abgelegt sind. Dazu dient das Modell eines mehrdimensionalen Datenwürfels. Im einfachen Fall bildet das Produkt (P_1, P_2, $\ldots$, P_n), der Verantwortungssbereich ($Nord, West, \ldots$) und die Zeit ($Quartal_1$, $Quartal_2$, $\ldots$) je eine Dimension des Würfels. Dieser Würfel prägt diejenigen betriebswirtschaftlichen Analysen, die ohne großen Aufwand durchführbar sind. Eine Schnitt durch den Würfel spannt eine Ebene auf, bei der eine Dimension einen vorgegebenen Wert hat. Beispielsweise wäre bei einer Analyse des verantwortlichen Produktmanagers $P_i = const$ während die anderen Dimensionen dann den Werterahmen bilden. Alle möglichen Schnitte (*Slices*) oder Würfelungen (*Dices*) bilden die Basis für die rasch ermittelbaren Analysen.

Datenwürfel

[43] Quelle: ↪[St+99] Seite 417.

2.5.2 *Data Mining*



Aufbauend auf dem *Data Warehouse* kann man statistische Verfahren nutzen, um mit Hilfe eines entsprechenden Analyseprogramms besondere Datenkonstellationen herauszufinden. Beispielsweise könnte ein solches Programm erkennen, daß ein nicht zufriedenstellendes Umsatzgesamtergebnis aufgrund des Produktes P_j im Zeitraum $Z_2 \ldots Z_7$ im Verantwortungsbereich $West$ verursacht wurde.

Unbekanntes schürfen

Eine solche Analyse, bei der ein bestimmtes Ziel vom Benutzer vorgegeben wird und das Programm selbst passende Beurteilungskriterien ermittelt und mit diesen die Daten überprüft, bezeichnet man als *Data Mining*[44]. Im riesigen „Datenberg“ wird mittels vorgegebener Methode „geschürft“ und man läßt sich überraschen was ans Tageslicht gefördert wird. Auf diese Art und Weise sollen bisher unbekannte Trends erkannt werden.

2.6 Übungen

2.6.1 Addition zweier Binärzahlen

Berechnen Sie $x + y$ mit $x = -14_{10}$ und $y = -12_{10}$ als Binärzahl (im Zweierkomplement).

2.6.2 Umformung eines Boolschen Ausdrucks

Prüfen Sie, ob sich die folgende Gleichung aus den Umformungsregeln für boolsche Ausdrücke ableiten läßt.
$\overline{(a \wedge b) \vee c} = (\overline{a} \vee \overline{b}) \wedge \overline{c}$

[44] Hinweis: Die deutsche Übersetzung „Daten-Bergbau“ ist unüblich ↪[Han96] S. 274.

2.6.3 Formale Sprache interpretieren

Die simple Sprache $\mathcal{SZP}$ (Shell zum Positionieren) dient zur Verschiebung und Einfärbung von graphischen Objekten auf einem Bildschirm. Zum Verschieben verfügt $\mathcal{SZP}$ über die vier Konstrukte („Verschiebekommandos“):

- 01 $\equiv$ verschieben in $-x$-Richtung, das heißt nach links.
- 10 $\equiv$ verschieben in $+x$-Richtung, das heißt nach rechts.
- 11 $\equiv$ verschieben in $+y$-Richtung, das heißt nach oben.
- 00 $\equiv$ verschieben in $-y$-Richtung, das heißt nach unten.

Zusätzlich verfügt $\mathcal{SZP}$ zum Einfärben oder zur Farberhaltung über die vier Konstrukte („Farbkommandos“):

- 01 $\equiv$ einfärben in Rot.
- 10 $\equiv$ einfärben in Blau.
- 11 $\equiv$ einfärben in Gelb.
- 00 $\equiv$ keine Farbänderung.

In einem $\mathcal{SZP}$-Programm folgt nach einem Verschiebekommando die Anzahl der Pixel, um die das Objekt zu verschieben ist. Diese Angabe ist konstant 6 Bit lang. Danach folgt stets ein Farbkommando in der Länge von 2 Bit. Der $\mathcal{SZP}$-Programmcode:

```
1100100001
0111111101
0111111101
0111111100
1010100001
1010100001
0011111100
0011111100
```

Das markierte Objekt FOO befindet sich vor der Ausführung des obigen $\mathcal{SZP}$-Programmcodes im Nullpunkt eines x, y-Koordinatensystem.

Farbe feststellen

Geben Sie die Farbe für FOO nach der Ausführung des obigen $\mathcal{SZP}$-Programmcodes an.

Position feststellen

Geben Sie den x, y-Wert für FOO nach der Ausführung des obigen $\mathcal{SZP}$-Programmcodes in Dezimalzahlen an.

2.6.4 Web-Publikation

Hypertext Markup Language (HTML) ist die Programmiersprache im World Wide Web (kurz: Web oder WWW). Im folgenden ist das HTML-Dokument `phasen.html` dargestellt.

`phasen.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
  "http://www.w3c.org/TR/REC-html40/strict.dtd">
<HTML lang="de">
<!--  Bonin: 24-Jun-1999                    -->
<!--  Update:14-Sep-1999                    -->
<HEAD>
<STYLE type="text/css">
  P.rechtsKlein {
    text-align: right;
    font-size: 6pt;
  }
  P {
    text-align: left;
    font-size: 12pt;
  }
  OL.gross {
    text-align: left;
    font-size: 20pt;
  }
  EM {
    font-size: 20pt;
    font-style: italic;
    color: #FFFFFF;
    background-color: teal;
  }
  H2 {
    text-align: center;
    font-size:  12pt;
  }
  H1 {
    text-align: center;
    font-size:  20pt;
```

```
33    }
34    BODY {
35      color: black;
36      background-color: #FFFFFF
37    }
38  </STYLE>
39    <TITLE>Software-Entwicklungskonzept</TITLE>
40  </HEAD>
41  <BODY>
42    <HR>
43    <H1>Vorgehensweise zur
44        <EM>Planung &amp; Realisierung</EM>
45        eines Softwaresystems</H2>
46    <H2>Klassische Phasenkonzept</H2>
47    <OL class="gross">
48      <LI>Phase:
49        <A HREF="planungsPhase.html">Planung</A>
50      <LI>Phase:
51        <A HREF="definitionsPhase.html">Definition</A>
52      <LI>Phase:
53        <A HREF="entwurfsPhase.html">Entwurf</A>
54      <LI>Phase:
55        <A HREF="implementationsPhase.html">Implementation
56      <LI>Phase:
57        <A HREF="abnahmeEinfuehrPhase.html">Abnahme-
58        &amp; Einf&uuml;hrung</A>
59      <LI>Phase:
60        <A HREF="wartungsPhase">Wartung</A>
61    </UL>
62    <HR>
63    <P class="rechtsKlein">
64      Copyright Bonin 24-Jun-1999 all rights reserved
65      <A HREF="http://as.fh-lueneburg.de/">Home</A>
66    </P>
67  </BODY>
68  <!-- Quelle: /u/bonin/mywww/ewi/ss99/phasen.html -->
69  </HTML>
70
```

HTML-Struktur überprüfen

Ein valides HTML-Dokument hat eine strikte Blockstruktur, das heißt, Konstrukte stehen in einer Sequenz oder sind völlständig ineinander geschachtelt. Das HTML-Dokument `phasen.html` ist nicht strikt blockstrukturiert. Beschreiben und korrigieren Sie die Verletzungen der Blockstruk-

tur.

CSS erläutern

Im HTML-Dokument `phasen.html` wird über das `<STYLE>`-Konstrukt das Konzept „*Cascading Style Sheets*“ (CSS) genutzt. Skizzieren Sie die Vorteile der CSS-Nutzung.

Interpretieren des korrigierten HTML-Dokumentes

Geben Sie das Ergebnis an, wenn Ihr korrigiertes HTML-Dokument `phasen.html` von einem Web-Browser angezeigt wird.

2.6.5 XML-Dokument mit DTD

Die folgende XML-Datei `HTMLSyn.xml` enthält zu Beginn eine interne *Document Type Definition* (DTD).

`HTMLSyn.xml`

```
 1  <?xml version="1.0" encoding="ISO-8859-1"?>
 2  <!--
 3
 4  Bonin: 05-Nov-1999 Update:
 5
 6  HTML-Syntaxbeispiel
 7
 8  XML-Notation des HTML-Konstruktes:
 9    <H1>Wirtschaftsinformatik macht viel
10      <A HREF="myserver.de/freude.html">Freude!</A>
11    </H1>
12
13  -->
14  <!DOCTYPE KONSTRUKT [
15  <!ELEMENT KONSTRUKT
16    (ANFANGSMARKE, (TEXT? |
17      (VORTEXT?,  KONSTRUKT, NACHTEXT?)*),
18      ENDMARKE?)>
19  <!ELEMENT ANFANGSMARKE (BEZEICHNER, ATTRIBUT*)>
20  <!ELEMENT BEZEICHNER (#PCDATA)>
21  <!ELEMENT ATTRIBUT (ATTRIBUTNAME, (GLEICH, VALUE)?)>
22  <!ELEMENT ATTRIBUTNAME (#PCDATA)>
23  <!ELEMENT GLEICH (#PCDATA)>
24  <!ELEMENT VALUE (#PCDATA)>
```

```
<!ELEMENT TEXT (#PCDATA)>
<!ELEMENT VORTEXT (#PCDATA)>
<!ELEMENT NACHTEXT (#PCDATA)>
<!ELEMENT ENDMARKE (#PCDATA)>
]>
<KONSTRUKT>
  <ANFANGSMARKE>
    <BEZEICHNER>
      H1
    </BEZEICHNER>
  </ANFANGSMARKE>
    <VORTEXT>
     Wirtschaftsinformatik macht viel
    </VORTEXT>
    <KONSTRUKT>
      <ANFANGSMARKE>
        <BEZEICHNER>
          A
        </BEZEICHNER>
        <ATTRIBUT>
          <ATTRIBUTNAME>
            HREF
          </ATTRIBUTNAME>
          <GLEICH>=</GLEICH>
          <VALUE>
            myserver.de/freude.html
          </VALUE>
        </ATTRIBUT>
      </ANFANGSMARKE>
      <TEXT>
        Freude!
      </TEXT>
      <ENDMARKE>
        /A
      </ENDMARKE>
    </KONSTRUKT>
  <ENDMARKE>
    /H1
  </ENDMARKE>
</KONSTRUKT>
<!-- End of object HTMLSyn.xml                -->

```

Dokument validieren

Prüfen Sie ob der Rest dieser Datei dieser internen DTD genügt.

Verbesserungsvorschlag für die DTD

Die DTD ist verbesserungsbedürftig. Machen Sie einen Vorschlag.

2.6.6 Datenhierarchie in HTML

In der folgenden HTML-Datei `divsingle.html` sind die Daten hierarchisch strukturiert. Der Inhalt dieses Dokumentes entspricht dem Inhalt der XML-Datei `wisingle.xml` ↪Seite 30.

`divsingle.html`

```
 1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
 2    "http://www.w3c.org/TR/REC-html40/strict.dtd">
 3  <!-- Bonin 09-Nov-1999                          -->
 4  <HTML lang="de">
 5  <HEAD>
 6  <TITLE>HTML-Hierarchie mit class-Attribut</TITLE>
 7  <STYLE type="text/css">
 8    DIV.WI {
 9      text-align: left;
10      font-size: 20pt;
11      color: #FFFFFF;
12    }
13    DIV.HARDWARE {
14      background-color: green;
15    }
16    DIV.SOFTWARE {
17      background-color: blue;
18    }
19    DIV.ORGANISATION {
20      background-color: red;
21    }
22    DIV.NUTZER {
23      background-color: black;
24    }
25    DIV.BETROFFENE {
26      color: #000000;
27      background-color: teal;
28    }
29  </STYLE>
30  </HEAD>
31  <BODY>
32  <H1>WI-(Denk-)Welt: Beispiel Einzelk&auml;mpfer</H1>
33  <DIV class="WI">
34    <DIV class="HARDWARE">
```

```
35      Notebook
36    </DIV>
37    <DIV class="SOFTWARE">
38      MS Windows NT 4.0,
39      MS Office
40    </DIV>
41    <DIV class="ORGANISATION">
42      Handwerksbetrieb
43    </DIV>
44    <DIV class="NUTZER">
45      DV-Freak
46    </DIV>
47    <DIV class="BETROFFENE">
48      Kollegen, Familie
49    </DIV>
50  </DIV>
51  </BODY>
52  <!-- End of object DIVsingle.html         -->
53  </HTML>
54
```

Dokument skizzieren

Skizzieren Sie die Darstellung der Datei `divsingle.html` in einem Standardbrowser; zum Beispiel in *Microsoft's Internet Explorer, Version 5*.

Vergleich: HTML- mit XML-Dokument

Vergleichen Sie das XML-Dokument `wisingle.xml` mit dem HTML-Dokument `divsingle.html` im Hinblick auf die Datenstrukturierung.

2.6.7 XML-Dokument `Manuscript.xml` analysieren

Im Rahmen eines *Content Management* Projektes ist die Datei `Manuscript.xml` zu speichern, allerdings nicht als CLOB (*Character Large Object*) in einer relationalen Datenbank (Beispiel: Oracle 9i) sondern entsprechend seiner rekursiven *Document Type Definition* in einer XML-Datenbank (Beispiel: Tamino 2.3.1). Beide Dateien sind im Folgenden angegeben.

Manuscript.dtd

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Bonin     21-Jan-2002              -->
<!ELEMENT Manuscript
  (Preface, Chapter+, Appendix)>
  <!ATTLIST Manuscript
    title   CDATA  #REQUIRED
    author  CDATA  #REQUIRED
    version CDATA  #IMPLIED>
<!ELEMENT Preface (Paragraph)+>
<!ELEMENT Chapter (Paragraph+ | Chapter+)>
  <!ATTLIST Chapter
    label   ID     #REQUIRED
    title   CDATA  #REQUIRED
    version CDATA  #IMPLIED>
<!ELEMENT Appendix (Paragraph+ | Chapter+)>
<!ELEMENT Paragraph (#PCDATA)>
  <!ATTLIST Paragraph
    label   ID     #REQUIRED
    version (final | draft) "draft"
    keywords CDATA #IMPLIED>
<!-- End of DTD: Manuscript.dtd          -->

```

Manuscript.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Manuscript Aspect-oriented Programming (AOP)
     Bonin   21-Jan-2002                  -->
<!DOCTYPE Manuscript SYSTEM "Manuscript.dtd">
<Manuscript
  title="Aspect-oriented Programming with Java"
  author="Hinrich E.G. Bonin"
  version="1.0">
  <Preface>
    <Paragraph label="Preface" version="final">
      Examples of post-object programmming technologies include
      domain-specific languages, generative programming,
      constraint languages, reflection and metaprogramming. ...
    </Paragraph>
  </Preface>
  <Chapter label="Intro" title="Introduction" version="draft">
    <Chapter label="Intro1" title="Structured Programming"
      version="final">
      <Paragraph label="Intro1.1">
        Key Concept: Explicit control constructs
      </Paragraph>
```

```
22        <Paragraph label="Intro1.2">
23          Constructs: Do, while and other loops, blocks, and so forth
24        </Paragraph>
25      </Chapter>
26      <Chapter label="Intro2" title="Modular Programming"
27        version="final">
28        <Paragraph label="Intro2.1">
29          Key Concept: Information hiding
30        </Paragraph>
31        <Paragraph label="Intro2.2">
32          Constructs: Modules with well-defined enforced interfaces
33        </Paragraph>
34      </Chapter>
35      <Chapter label="Intro3" title="Data Abstraction" version="final">
36        <Paragraph label="Intro3.1">
37          Key Concept: Hide the representation of data
38        </Paragraph>
39        <Paragraph label="Intro3.2">
40          Constructs: Types
41        </Paragraph>
42      </Chapter>
43      <Chapter label="Intro4" title="Object-oriented Programming"
44        version="final">
45        <Paragraph label="Intro4.1">
46          Key Concept: Objects, with classification and specialization
47        </Paragraph>
48        <Paragraph label="Intro4.2">
49          Constructs: Classes, objects, polymorphism
50        </Paragraph>
51      </Chapter>
52    </Chapter>
53    <Chapter label="AOP" title="Adaptive Methods" version="draft">
54      <Paragraph label="AOP.DJ-Library">
55        An operation in an object-orieted program often involves several
56        different collaborating classes. ...
57      </Paragraph>
58      <Paragraph label="AOP.aspectJ" version="outline">
59        An aspect-oriented extension to Java enables plug-and-play
60        implementations of crosscutting.
61      </Paragraph>
62    </Chapter>
63    <Appendix>
64      <Chapter  title="References">
65        <Paragraph label="Journals" version="draft">
66          Communications of the ACM October 2001/Vol.44, No.10
67        </Paragraph>
```

```
68        <Paragraph label="Links" version="draft">
69          http://aspectj.org visited 21-Jan-2002
70        </Paragraph>
71      </Chapter>
72    </Appendix>
73  </Manuscript>
74  <!-- End of object Manuscript.xml  -->
75
```

Well-formed-**Prüfung**

Stellen Sie fest, ob die Datei `Manuscript.xml` mit dem Browser *Microsoft Internet Explorer 5* angezeigt und ausgedruckt werden kann. [Hinweis: Dieser Browser prüft nur ob das Dokument *well-formed* ist.]

Dokument validieren

Stellen Sie fest, ob die Datei `Manuscript.xml` seiner *Document Type Definition* `Manuscript.dtd` entspricht. Wenn Sie Fehler erkennen, dann erläutern und korrigieren Sie diese.

2.6.8 XML-Dokument `RubyScripting.xml` analysieren

Das innovative Softwareunternehmen eL GmbH, Hamburg (*eLearning*) nutzt für elektronische Dokumente eine *Document Type Definition*, die in der Datei `Manual.dtd` festgelegt ist. Der Mitarbeiter Franz Schlaukopf hat begonnen die Software *Ruby*, eine objekt-orientierte Scriptsprache, in der Datei `RubyScripting.xml` zu beschreiben. Beide Dateien sind im Folgenden angegeben.

`Manual.dtd`

```
 1  <?xml version="1.0" encoding="ISO-8859-1"?>
 2  <!-- eL GmbH   19-Jan-2004              -->
 3  <!ELEMENT Manual
 4    (Abstract ,Preface, Chapter+, Appendix)>
 5    <!ATTLIST Manual
 6      title   CDATA  #REQUIRED
 7      author  CDATA  #REQUIRED
 8      version CDATA  #IMPLIED>
 9  <!ELEMENT Abstract (Paragraph)?>
10  <!ELEMENT Preface (Paragraph)+>
```

```
11  <!ELEMENT Chapter (Paragraph+ | Chapter+)>
12    <!ATTLIST Chapter
13      label   ID     #REQUIRED
14      title   CDATA  #REQUIRED
15      version CDATA  #IMPLIED>
16  <!ELEMENT Appendix (Paragraph+ | Chapter+)>
17  <!ELEMENT Paragraph (#PCDATA)>
18    <!ATTLIST Paragraph
19      label   ID     #REQUIRED
20      version (final | draft) "draft"
21      keywords CDATA #IMPLIED>
22  <!-- End of DTD: Manual.dtd            -->
23
```

RubyScripting.xml

```
 1  <?xml version="1.0" encoding="ISO-8859-1"?>
 2  <!-- Manual Ruby Scripting
 3       eL GmbH 19-Jan-2004                  -->
 4  <!DOCTYPE Manual SYSTEM "Manual.dtd">
 5  <Manual
 6    title="Thinking in Ruby"
 7    author="Franz Schlaukopf, eL GmbH"
 8    version="1.0">
 9    <Abstract>
10    </Abstract>
11    <Preface>
12      <Paragraph label="Preface" version="final">
13        Ruby is the interpreted scripting language for quick
14        and easy object-oriented programming. It has many
15        features to process text files and to do system
16        management tasks.
17      </Paragraph>
18    </Preface>
19    <Chapter label="Intro" title="Introduction" version="draft">
20      <Chapter label="Intro1" title="Features of Ruby"
21        version="final">
22        <Paragraph label="Intro1.1">
23          Simple Syntax
24        </Paragraph>
25        <Paragraph label="Intro1.2">
26          Normal and advanced object-oriented features
27          (ex. class, method calls, mix-in, singleton-method)
28        </Paragraph>
29        <Paragraph label="Intro1.3">
30          Operator Overloading
31        </Paragraph>
```

```
32        <Paragraph label="Intro1.4">
33          Exception Handling
34        </Paragraph>
35        <Paragraph label="Intro1.5">
36          Garbage Collection
37        </Paragraph>
38        <Paragraph label="Intro1.6">
39          Dynamic Operator Overloading
40        </Paragraph>
41      </Chapter>
42      <Chapter label="Intro2" title="Highly Portable"
43        version="final">
44        <Paragraph label="Intro2.1">
45          Ruby works on many UNIX Maschines, and on DOS,
46          Windows, Mac, BeOS etc.
47        </Paragraph>
48      </Chapter>
49      <Chapter label="Intro3" title="Notation Conventions"
50        version="final">
51        <Paragraph label="Intro3.1">
52          Ruby program files use the suffix *.rb.
53          If you don't want a console window to be created,
54          use the suffix *.rbw instead.
55        </Paragraph>
56        <Paragraph label="Intro3.2">
57           Program files may be created using any plain
58           text editor such as Emacs, jEdit or Eclipse.
59        </Paragraph>
60      </Chapter>
61    </Chapter>
62    <Chapter label="RI" title="Ruby Information at your fingertips"
63      version="draft">
64      <Paragraph label="RI.call">
65        ri displays documentation for the named classes
66        or methods. All names can be abbreviated to their
67        minimum non-ambiguous size.
68      </Paragraph>
69      <Paragraph label="RI.copyright" version="drafty">
70        The program files ri and refdoc.rb are copyright
71        (c) 2001 Dave Thomas, The Pragmatic Programmers,
72        and are released under the same terms as Ruby.
73      </Paragraph>
74    </Chapter>
75    <Appendix>
76      <Chapter  title="References">
77        <Paragraph label="Book.01" version="draft">
```

```
78          David Thomas / Andrew Hunt; Programmieren mit Ruby,
79          deutsche Übersetzung von Reder Translations, München u.a.
80          (Addison-Wesley), 2002, ISBN 3-8273-1965-X (german).
81        </Paragraph>
82        <Paragraph label="Link.A" version="draft">
83          http://www.ruby-lang.org/en/ visited 28-Nov-2003
84        </Paragraph>
85      </Chapter>
86    </Appendix>
87  </Manual>
88  <!-- End of object RubyScripting.xml  -->
89
```

Well-formed-**Prüfung**

Stellen Sie fest, ob die Datei `RubyScripting.xml` mit dem Browser *Microsoft Internet Explorer 6* angezeigt und ausgedruckt werden kann. [Hinweis: Dieser Browser prüft nur ob das Dokument *well-formed* ist.]

Dokument validieren

Stellen Sie fest, ob die Datei `RubyScripting.xml` ihrer *Document Type Definition* `Manual.dtd` entspricht. Wenn Sie Fehler erkennen, dann erläutern und korrigieren Sie diese. Geben Sie dabei die betroffenen Zeilennummern an.

2.6.9 Web-Page für *Jung & Müller* erstellen

Die Glashütte *Jung & Müller AG, Lüneburg* stellt Spezialgläser für die Restaurierung von alten Kirchenfenstern her. Ihre Home-Page hat die Adresse `http://jung-mueller.de`. Sie arbeitet zur Zeit für vier Restaurationsbetriebe. Diese haben die folgenden Web-Adressen:

- KirchenStudio GmbH, Magdeburg: `http://kirchen-studio.de`
- Glaserei Otto AG, Bremen: `http://glaserei-otto.com`
- Glas & Restauration AG, Freiburg: `http://glas-restauration.de`
- Alte Gläser GmbH, Karlsruhe: `http://alte-glaeser.de`

Web-Master der *Jung & Müller AG, Lüneburg* ist Herr Norbert Schmitt. Er hat die email-Adresse `schmitt@jung-mueller.de`.

XHTML-Dokument codieren

Codieren Sie ein striktes XHTML-Dokument. Benutzen Sie dazu die Datei `partner.html`. Die von Ihnen vervollständigte Datei `partner.html` soll:

1. als eine geordnete Liste die Links zu den vier Geschäftspartnern und
2. in der Hauptüberschrift den Link zur eigenen Home-Page aufweisen.
3. Zusätzlich soll sie einen email-Kontakt zum Web-Master enthalten.

`partner.html`

```
 1  <?xml version="1.0" encoding="utf-8" ?>
 2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 3    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
 4  <!-- Jung & Mueller 19-Jan-2004  -->
 5  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de">
 6  <head>
 7  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
 8  <meta name="robots" content="index,follow" />
 9  <link href="partner.css" rel="stylesheet" type="text/css" />
10  <title>Gesch&auml;ftspartner der Jung &amp; M&uuml;ller AG</title>
11  </head>
12  <body>
13  <h1 class="large">Gesch&auml;ftspartner der</h1>
14  <h1 class="large">
15
16  </h1>
17  <   class="normal">
18    <li>Partner:
19
20
21    <li>Partner:
22
23
24    <li>Partner:
25
26
27    <li>Partner:
28
```

```
29
30  </  >
31  <p class="small">Bei Fragen wenden Sie sich bitte an
32  Web-Master
33
34
35  </body>
36  </html>
37
```

CSS-Datei vervollständigen

Vervollständigen Sie die CSS-Datei (Cascading Style Sheet) `partner.css`. Die von Ihnen vervollständigte Datei `partner.css` soll den Hinweis zum Web-Master in gelber Schrift auf schwarzem Hintergrund darstellen.

`partner.css`

```
 1  /* Cascading Style Sheet: meyer.css  */
 2  /* Jung & Mueller AG    19-Jan-2004  */
 3        {
 4    font-size:  10pt;
 5    color:      yellow;
 6    background: black;
 7  }
 8  .normal {
 9    font-size:  14pt;
10  }
11  .large {
12    font-size:  24pt;
13  }
14  body {
15    color:      black;
16    background: yellow;
17  }
18
```

CSS-Konstruktion erläutern

Skizzieren Sie Vorteile und Nachteile der CSS-Konstruktion.

2.6.10 Web-Page für *Wundort* erstellen

Die kleine Gemeinde *Wundort* im Landkreis *Waldberg* beabsichtigt im Rahmen der Landkreisaktion „Alle rein ins Internet“ auch eine Page ins Web zu stellen. Vom Landkreis erhält Sie dazu folgende Adressen:

- für den Bürgermeister:
  `http://chef.wundort.waldberg.de`
- für den Fachbereich Ordnungswesen:
  `mailto:ordnung@verwaltung.wundort.waldberg.de`
- für den Fachbereich Finanzwesen:
  `mailto:finanzen@verwaltung.wundort.waldberg.de`
- und für den Web-Beauftragten:
  `mailto:web@wundort.waldberg.de`

Derzeit ist Herr Herbert Rege Bürgermeister von *Wundort*. Er ist über die Telefonnummern 04131/6666 oder 0166666666 direkt erreichbar. Dem Fachbereich Ordnungswesen steht Amtsleiter Franz Fuchs und dem Fachbereich Finanzwesen Kämmerer Willi Armmaus vor. Web-Beauftragter ist Amtsrat Karl Meyer.

Der Landkreis *Waldberg* hat die Adresse `http://waldberg.de`.

XHTML-Dokument codieren

Codieren Sie ein striktes XHTML-Dokument. Benutzen Sie dazu die Datei `wundort.html`. Die von Ihnen vervollständigte Datei `wundort.html` soll:

1. als eine geordnete Liste die Links (Web bzw. Mail) zum Bürgermeister, zu den beiden Fachbereichen und
2. in der Hauptüberschrift den Link zum Landkreis aufweisen.
3. Zusätzlich soll der Mailkontakt zum Web-Beauftragten enthalten sein.

Der Bürgermeister Herbert Rege ist ein erfahrener Kommunalpolitiker. Im Sinne von mehr Bürgerfreundlichkeit möchte er auch die jeweilgen Vor- und Zunamen ausgewiesen haben.

XHMTL-Datei `wundort.html`

```
 1  <?xml version="1.0" encoding="utf-8" ?>
 2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 3    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
 4  <!-- Gemeinde Wundort 28-Jun-2004  -->
 5  <html xmlns="http://www.w3.org/1999/xhtml"
 6    xml:lang="de">
 7  <head>
 8  <meta http-equiv="Content-Type"
 9    content="text/html; charset=utf-8" />
10  <meta name="robots" content="index,follow" />
11  <link href="waldberg.css" rel="stylesheet"
12    type="text/css" />
13  <title>
14  </head>
15  <body>
16  <h1 class="large">Gemeinde Wundort im Landkreis
17
18  <h1 class="large">Verwaltung</h1>
19  <ol class="normal">
20    <li>
21    <a href="http://chef.wundort.waldberg.de">
22      Herbert Rege</a><br />
23      Tel.: 04131/6666 oder 0166666666</li>
24    <li>
25
26    </li>
27    <li>
28
29
30
31  <p class="small">Bei Fragen wenden Sie sich
32    bitte an den Web-Beauftragten
33    <a href="mailto:web@wundort.waldberg.de">
34
35
36  </body>
37  </html>
38
```

Farbmodifikation

Die Gemeinde *Wundort* ist verpflichtet die CSS-Datei (Cascading Style Sheet) `waldberg.css` zu nutzen. Das schreibt der Landkreis vor.

CSS-Datei `waldberg.css`

```
 1  /* Cascading Style Sheet: waldberg.css */
 2  /* Landkreis Waldberg  28-Jun-2004  */
 3  .small {
 4    font-size:  10pt;
 5    color:      blue;
 6    background: white;
 7  }
 8  .normal {
 9    font-size:  14pt;
10  }
11  .large {
12    font-size:  24pt;
13  }
14  body {
15    color:      white;
16    background: black;
17  }
18
```

Der Bürgermeister Herbert Rege ist einziger Bürgermeister im Landkreis *Waldberg*, der FDP-Mitglied ist. Er möchte daher auf seiner Web-Page im Regelfall einen gelben Hintergrund haben. Die Schrift sollte in blauer Farbe sein. Geben Sie die zusätzlich benötigten Zeilen in der Datei `wundort.html` im Folgenden an. Nennen Sie die Zeilennummer nach der Ihr zusätzlicher Code einzufügen ist.

WCMS: Vor- & Nachteile

Skizzieren Sie Vorteile und Nachteile eines WCMS (*Web-Content-Management-System* — zum Beispiel Typo3 —) im Vergleich zur Erstellung der Web-Pages ohne Werkzeugunterstützung.

2.6.11 XML-Dokument `Modellierung.xml` analysieren

Das Beratungsunternehmen eGAG, München (*electronic Government AG*) mit dem Arbeitsschwerpunkt im öffentlichen Sektor nutzt für die Kooperation und Kommunikation mit Experten unterschiedlicher Herkunft eine *Document Type Definition*, die in der Datei `Modellierung.dtd` festgelegt ist. Die derzeitige Ausprägung ihres „vier Säulenkonzeptes“ zur Modellierung ist in der Datei `Modellierung.xml` beschrieben. Beide Dateien sind im Folgenden angegeben.

Modellierung.dtd

```
 1  <?xml version="1.0" encoding="ISO-8859-1"?>
 2  <!-- eGAG   28-Jun-2004                -->
 3  <!ELEMENT Modellierung
 4    (Pragmatik, Formalismus, Methodik, Werkzeug)>
 5    <!ATTLIST Modellierung
 6      modellart (Domäne | System) "Domäne"
 7      perspektive CDATA #REQUIRED>
 8  <!ELEMENT Pragmatik (Paragraph)+>
 9  <!ELEMENT Formalismus (Paragraph)+>
10  <!ELEMENT Methodik (Paragraph)+>
11  <!ELEMENT Werkzeug (Paragraph)+>
12  <!ELEMENT Paragraph (#PCDATA)>
13    <!ATTLIST Paragraph
14      label ID #REQUIRED
15      version (Abgenommen | Entwurf) "Entwurf"
16      keywords CDATA #IMPLIED>
17  <!-- End of DTD: Modellierung.dtd             -->
18
```

Modellierung.xml

```
 1  <?xml version="1.0" encoding="ISO-8859-1"?>
 2  <!-- Grundlagen der Kooperation bei der Modellierung
 3       eGAG 28-Jun-2004                   -->
 4  <!DOCTYPE Modellierung SYSTEM "Modellierung.dtd">
 5  <Modellierung
 6    modellart="System"
 7    perspektive="Kooperation">
 8  <Pragmatik>
 9  <Paragraph label="p1"
10    keywords="Beschreibungstechniken"
11    version="Entwurf">
12  In der Softwaretechnik haben sich eine Reihe
13  von anschaulichen Beschreibungstechniken,
14  insbesondere diagrammatischer Art, herausgebildet.
15  </Paragraph>
16  <Paragraph label="p2"
17    keywords="Boxologie"
18    version="Entwurf">
19  Die vielfältigen Diagramme werden auch als Boxologie
20  bezeichnet.
21  </Paragraph>
22  </Pragmatik>
23  <Formalismus>
24  <Paragraph label="f1"
25    keywords="Mathematik" version="Entwurf">
```

```
26  Wie üblich im Ingenieurbereich sind die Aneignung
27  und die Anwendung der einschlägigen Zweige der
28  Mathematik zwingend notwendig.
29  </Paragraph>
30  <Paragraph label="f2"
31    keywords="Präzision, Kosten, Nutzen"
32    version="Entwurf">
33  Die Mathematik gewährleitet Präzision und
34  Zuverlässigkeit. Sie hilft bei der Kostenbeherrschung
35  und der Nutzenabschätzung.
36  </Paragraph>
37  <Methodik>
38  <Paragraph label="m1"
39    keywords="Abstraktion"
40    version="Entwurf">
41  Die Abstraktion ist die Haupttechnik, um die
42  Komplexität von Software zu meistern.
43  </Paragraph>
44  <Paragraph label="m1"
45    keywords="Approximation"
46    version="Entwurf">
47  Die Abstraktion spielt eine ähnlich vereinfachende
48  Rolle wie die Approximation in anderen
49  Ingenieurbereichen.
50  </Paragraph>
51  </Methodik>
52  <Werkzeug>
53  <Paragraph label="w1"
54    keywords="CASE"
55    version="Fertig">
56  Methodisches Vorgehen erfordert in der Regel
57  den Einsatz von Spezialsoftware. Diese
58  bezeichnet man als CASE-Werkzeuge
59  (Computer Aided Software Engineering Tools).
60  </Paragraph>
61  </Werkzeug>
62  </Modellierung>
63  <!-- End of object Modellierung.xml  -->
64
```

Well-formed-Prüfung

Stellen Sie fest, ob die Datei `Modellierung.xml` mit dem Browser *Microsoft Internet Explorer* (ab Version 6), angezeigt und ausgedruckt werden kann. Begründen Sie Ihre Feststellung. [Hinweis: Dieser Brow-

ser prüft nur ob das Dokument *well-formed* ist.]

Dokument validieren

Stellen Sie fest, ob die Datei `Modellierung.xml` ihrer *Document Type Definition* `Modellierung.dtd` entspricht. Wenn Sie Fehler erkennen, dann erläutern und korrigieren Sie diese. Geben Sie dabei die betroffenen Zeilennummern an.

Kapitel 3

Rechnen: Steuerung von Abläufen

In welcher Reihenfolge sind Aufgaben, beispielsweise die Aufgabe $\mathcal{X}$ und die Aufgabe $\mathcal{Y}$, zu erledigen? Bedeutsam ist die Festlegung, ob erst $\mathcal{X}$ und danach $\mathcal{Y}$ durchzuführen ist oder ob beide voneinander unabhängig sind und daher nebenläufig durchführt werden können. Zur Steuerung solcher Abläufe erläutern wir die Notation der elementaren Konstrukte: Sequenz, Alternative, Iteration und Nebenläufigkeit. Dabei vergleichen wir die Darstellungen in Form des Programmablaufplanes, des Struktogrammes, der Entscheidungstabelle und der Programmiersprache Java^{TM}.

Wegweiser

Das Kapitel „Rechnen: Steuerung von Abläufen“ erläutert:

- elementare Konstrukte zur Ablaufsteuerung und
  ↪ Seite 144 . . .

- die Interpretation von Kommandos.
  ↪ Seite 159 . . .

3.1 Elementare Konstrukte zur Ablaufsteuerung

Luat eienr Stduie der Cambrdige Unievrstiat speilt es kenie Rlloe in welcehr Reiehnfogle die Buhcstbaen in eniem Wrot vorkmomen, die eingzie whctige Sahce ist, dsas der ertse und der lettze Buhcstbaen stmimt. Der Rset knan in eienm volilegen Duchrienanedr sein und knan trtozedm prboelmols gelseen wreden. Das ist, weil das menchsilche Ague nicht jeedn Buhcstbaen liset. Ertsuanlcih, nihc.

Im Gegensatz zur Lesestudie ist die Folge von Abarbeitungsschritten in einem Programm bedeutsam. Sie wird als Ablaufstruktur oder Kontrollstruktur bezeichnet. Eine Kontrollstrukturen setzt sich aus elementaren Konstrukten (*Primitives*) zusammen. Primitives[1] sind zum Beispiel:

1. die **Sequenz**,
2. die **Alternative**,
3. die **Iteration** und
4. die **Nebenläufigkeit** (*concurrency*).

Eine Kontrollstruktur läßt sich auf viele Arten notieren, beispielsweise als Programmablaufplan (PAP), Struktogramm (*Nassi/Shneiderman diagram* (NDS)), Entscheidungstabelle (ET) (*decision (structure) table*), Pseudocode (*process design language* (PDL)) oder direkt in einer Programmiersprache. Im Hinblick auf die Transparenz der Dokumentation hat jede Notation ihre Stärken und Schwächen. Aus der großen Menge mehr oder weniger formaler Darstellungsmittel wurden zur Erörterung der Primitives ausgewählt:

- der Programmablaufplan (PAP)
- das Struktogramm (*Nassi/Shneiderman diagram* (NSD)) ↪[Nas+73, DIN66261],
- die Entscheidungstabelle (ET) ↪[DIN66241, Str77] und
- die Programmiersprache Java^{TM}.

Diese Wahl berücksichtigt einerseits den Verbreitungsgrad und zeigt andererseits die Vielfalt der Darstellungsmittel.

[1] Hier ist keine Vollständigkeit angestrebt. Es gibt weitere relevante Konstrukte, zum Beispiel die **Rekursion** und die **Selbstbezüglichkeit** (z.B. Funktion kann sich selbst als Argument haben; Erläuterungen dazu zum Beispiel ↪[Bo91b]).

Tabellenname		$R_1, R_2, \ldots, R_n$
B_1	Bedingungsteil	Bedingungsanzeigerteil
B_2		
$\vdots$		
B_k		
A_1	Aktionsteil	Aktionsanzeigerteil
A_2		
$\vdots$		
A_m		

Legende:

$B_1, \ldots, B_k$ $\equiv$ Bedingungsbezeichner
$A_1, \ldots, A_m$ $\equiv$ Aktionsbezeichner
$R_1, \ldots, R_n$ $\equiv$ Regelbezeichner

Tabelle 3.1: Komponenten einer Entscheidungstabelle

Entscheidungstabelle (ET)

Im Rahmen einer Analyse sind Entscheidungstabellen[2] vorteilhaft bei Situationen (Sachverhalten), bei denen eindeutig definierbare Bedingungen zu eindeutig definierbaren Aktionen führen. Eine ET dokumen-

[2] In der Mitte der fünfziger Jahre entwickelte man quasi gleichzeitig bei General Electric Company *Decision Structure Tables* und bei Sutherland Company *Management Rules*.

1959: Erste Analysen und Dokumentationen mittels ET.

1960: Entwicklung der ET-Sprache TABSOL.

1962: Spezifikation der ET-Sprache DETAB-X. Sie sollte in COBOL-61 integriert werden. Dies gelang jedoch nicht.

1963: ET-Precompiler für COBOL

heute: ET-Technik ist Bestandteil von Werkzeugen zur Softwareerstellung.

tiert:

- die Voraussetzungen (*Bedingungen*), mit denen festgestellt wird, welcher Entscheidungsfall vorliegt, und
- die Handlungen (*Aktionen*), die in diesem Fall zu vollziehen sind.

Für die ET gibt es eine normierte Notation (↪[DIN66241]). Zur besseren Referenzierung wird in WI>Datadiese durch Bezeichner für Bedingungen und Aktionen ergänzt (↪Tabelle 3.1 Seite 145). Der Zwang, den Sachverhalt in Bedingungen und Aktionen einzuteilen, unterstützt den Prozeß der Präzisierung einzelner Beschreibungen und Aussagen. Die tabellarische Anordnung und die Analysmöglichkeiten der dokumentierten Fälle im Hinblick auf (syntaktische) Vollständigkeit und (formale) Widerspruchsfreiheit schaffen mehr Transparenz (Durchsichtigkeit, Klarheit) über die darzustellende Situation. Beim Grundaufbau einer ET gilt:

- Zwischen den einzelnen Bedingungen einer Bedingungsfolge $(B_1, \ldots, B_k)$ besteht eine logische UND-Beziehung.
- Zwischen den einzelnen Aktionen einer Aktionsfolge $(A_1, \ldots, A_m)$ besteht eine logische UND-Beziehung.
- Eine Bedingungsfolge steht zur zugehörigen Aktionsfolge (gleiche Spalte) in einer $WENN \ldots DANN \ldots$ Beziehung, das heißt, trifft eine Bedingungsfolge zu, dann ist die entsprechende Folge von Aktionen auszuführen.

Eintreffer-ET

- Bei der ET vom Typ *Eintreffer-ET* sind die Regeln $(R_1, \ldots, R_n)$ in einem exklusiven ODER-Bezug, das heißt, trifft eine Regel zu, dann trifft keine andere Regel mehr zu.[3]

Mehrtreffer-ET

- Bei der ET vom Typ *Mehrtreffer-ET* ist nach dem Zutreffen einer Regel zu prüfen, ob eine weitere Regel zutrifft. Ist dies der Fall, dann ist auch diese auszuführen. Die Richtung der Regelabarbeitung ist bei einer Mehrtreffer-ET vorab festzulegen. Üblicherweise ist eine Folge „von-links-nach-rechts“ definiert. Trifft eine Regel R_j zu, dann ist nach Vollzug die Prüfung bei Regel R_{j+1} fortzusetzen.[4]

[3] Vorausgesetzt die ET ist „korrekt“ , das heißt hier, sie ist ohne Redundanz.

[4] Näheres zur Mehrtreffer-ET zum Beispiel ↪[Str77].

3.1.1 Sequenz

Eine Sequenz ist in bezug auf ihre Verstehbarkeit eine sehr einfache Struktur. Eine semigraphische Notation ist daher häufig nicht notwendig.

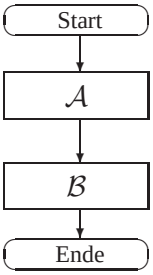
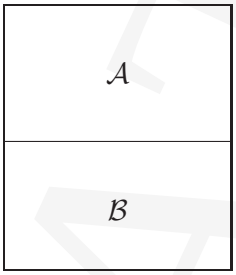
Quellcodebeispiel `Sequenz.java`

```
 1  /**
 2   Java-Beispiel zum Konstrukt: Alternative
 3    Bonin 15-Sep-1999
 4  */
 5
 6  public class Sequenz {
 7
 8    public static void main(String argv[]) {
 9
10      System.out.println("Run A");
11      System.out.println("Run B");
12    }
13  }
14  // End of file: /u/bonin/mywww/ewi/widata/Sequenz.java
15
```

Aufruf der Klasse `Sequenz.class`

```
>javac Sequenz.java
>java Sequenz
Run A
Run B
>
```

ET Bei einer ET können zwei Prozesse (Aufgaben, Vorgänge etc.) $\mathcal{A}$ und $\mathcal{B}$ als Sequenz durch Rückgriff auf eine Notation ohne Bedingungen abgebildet werden. Da eine begrenzte Eintreffer-ET bei n Bedingungen 2^n Regeln hat, ist eine Regel ($2^0 = 1$) bei dieser „entartete“ ET zu notierten (↪Tabelle 3.2 auf Seite 148).

Programmablaufplan ....pap[1]	Struktogramm ....nsd[2]
Sequenz.pap  	**Sequenz.nsd** 
Entscheidungstabelle ....et[3]	**Programmiersprache ....java[4]**

Sequenz.et | R1
A1 | $\mathcal{A}$ | X
A2 | $\mathcal{B}$ | X
	Sequenz.java
	`class Seq{`
	$\mathcal{A}$;
	$\mathcal{B}$;
	`}`

Legende:

[1] ≡ Programmablaufplan; auch: Ablaufplan, Ablaufdiagramm, Flußdiagramm
[2] ≡ Struktogramm (*Nassi/Shneiderman diagram*)
[3] ≡ Entscheidungstabelle (*decision (structure) table*)
[4] ≡ Programmiersprache Java^{TM}

Tabelle 3.2: Sequenz der Konstrukte $\mathcal{A}$ und $\mathcal{B}$

3.1.2 Alternative

if. . .then. . .else. . .-**Kontstrukt**

Die Alternative dient zum Verzweigen in alternative Programmblöcke. Dazu ist ein *Boolean-Ausdruck* (Prädikat) im Hinblick auf seinen Wahrheitswert auszuwerten. Der ermittelte Wert muß entweder als *true* oder als *false* interpretierbar sein. Bei einer Alternative muß erkennbar sein, wo das Prädikat aufhört und wo der *true*-Zweig beginnt. Auch die Abgrenzung des *true*-Zweiges vom *false*-Zweig muß eindeutig sein. Diese Unterscheidungen werden in Spezifikations- und/oder Programmiersprachen häufig durch Klammern verdeutlicht. (↪Tabelle 3.3 auf Seite 151).

Prädikat

Quellcodebeispiel `Alternative.java`

```
1  /**
2   Java-Beispiel zum Konstrukt: Alternative
3    Bonin 15-Sep-1999
4  */
5
6  public class Alternative {
7
8    static final boolean p = false;
9
10   public static void main(String argv[]) {
11
12     if (p) {
13         System.out.println("Run A");
14       }
15       else {
16         System.out.println("Run B");
17       }
18   }
19 }
20 // End of file: /u/bonin/mywww/ewi/widata/Alternative.java
21
```

Aufruf der Klasse `Alternative.class`

```
>javac Alternative.java
>java Alternative
```

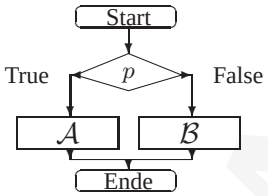
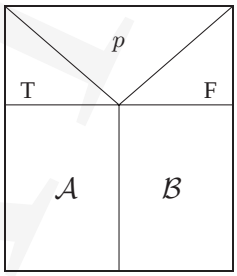
```
Run B
>
```

case-Konstrukt

Die ET ist gut geeignet geschachtelte Alternativen, Fallunterscheidungen (≡ *case*-Konstrukt), darzustellen, insbesondere wenn sie konsolidiert werden kann (↪Tabelle 3.4 auf Seite 153).

Quellcodebeispiel `Case.java`

```
 1  /**
 2   Java-Beispiel zum Konstrukt: Case
 3    Bonin 15-Sep-1999
 4  */
 5
 6  public class Case {
 7
 8    static final int x = 1;
 9    static final int y = 2;
10    static final int z = 3;
11
12    public static void main(String argv[]) {
13      Integer i = new Integer(argv[0]);
14      int v = i.intValue();
15      System.out.println("Aufruf mit " +  v + ".");
16
17      switch (v) {
18        case x :
19          System.out.println("Run A");
20          break;
21        case y :
22          System.out.println("Run B");
23          break;
24        case z :
25          System.out.println("Run C");
26          break;
27        default:
28          System.out.println("Run D");
29      }
30    }
31  }
```

Programmablaufplan ....pap	Struktogramm ....nsd
Alternative.pap  	**Alternative.nsd** 
Entscheidungstabelle ....et	**Programmiersprache ....java**

Alternative.et | R1 | R2
B1 | p | J | N
A1 | $\mathcal{A}$ | X |
A2 | $\mathcal{B}$ | | X
	Alternative.java
	`class Alternative{`
	⋮
	`if (`p`)`
	$\mathcal{A}$;
	`else`
	$\mathcal{B}$;
	}

Legende:
↪Tabelle 3.2 auf Seite 148

Tabelle 3.3: Alternative mit dem Prädikat p und den Konstrukten $\mathcal{A}$ und $\mathcal{B}$

```
32  // End of file: /u/bonin/mywww/ewi/widata/Case.java
33
```

Aufruf der Klasse `Case.class`

```
>javac Case.java
>java Case 3
Aufruf mit 3.
Run C
>java Case 5
Aufruf mit 5.
Run D
>
```

3.1.3 Iteration

Die ET benötigt zur Notation einer Iteration ihren *Selbstaufruf*.

while-Konstrukt

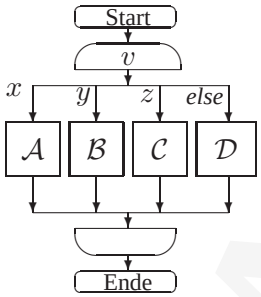
Beim *while*-Konstrukt ist das Struktogramm einfach zu überschauen (↪Tabelle 3.5 auf Seite 155). Der PAP ist vergleichsweise weniger übersichtlich. Die ET verlangt ein Lesen der Aktionen und dabei ein Herausfinden der entsprechenden *run*-Angabe.

Quellcodebeispiel `While.java`

```
 1  /**
 2    Java-Beispiel zum Konstrukt: While
 3    Bonin 15-Sep-1999
 4  */
 5
 6  public class While {
 7
 8    public static void main(String argv[]) {
 9      Integer i = new Integer(argv[0]);
10      int v = i.intValue();
11      System.out.println("Aufruf mit " +  v + ".");
12
13      while (v > 0) {
14          System.out.println("Run A");
```

Programmablaufplan . . ..pap

Case.pap



Struktogramm . . ..nsd

CASE.nsd

x	y	z	*else*
$\mathcal{A}$	$\mathcal{B}$	$\mathcal{C}$	$\mathcal{D}$

Entscheidungstabelle . . ..et

Case.et		R1	R2	R3	R4
B1	$v = x$	J	N	N	N
B2	$v = y$	–	J	N	N
B3	$v = z$	–	–	J	N
A1	$\mathcal{A}$	X			
A2	$\mathcal{B}$		X		
A3	$\mathcal{C}$			X	
A4	$\mathcal{D}$				X

Programmiersprache . . ..java

Case.java

```
class Case{
⋮
   switch (v){
   case x:
      A;
      break;
   case y:
      B;
      break;
   case z:
      C;
      break;
   default:
      D;
   }
}
```

Legende:
↪Tabelle 3.2 auf Seite 148

Tabelle 3.4: *Case*-Konstrukt mit der Variable v und ihren Werten x, y und z

```
15          v = v - 1;
16      }
17    }
18  }
19  // End of file: /u/bonin/mywww/ewi/widata/While.java
20
```

Aufruf der Klasse `While.class`

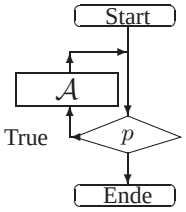
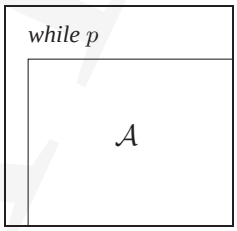
```
>javac While.java
>java While 3
Aufruf mit 3.
Run A
Run A
Run A
>java While 0
Aufruf mit 0.
>
```

until-Konstrukt

Die ET kann ein *until*-Konstrukt nicht abbilden. Es bedarf einer Ersatznotation mit dem *while*-Konstrukt (↪Tabelle 3.6 auf Seite 157).

Quellcodebeispiel `Until.java`

```
 1  /**
 2   Java-Beispiel zum Konstrukt: Until
 3    Bonin 15-Sep-1999
 4
 5    Achtung!
 6
 7      Until-Ablauf anders als im Struktogramm:
 8       1. Kennwort while
 9       2. Iterationsende bei false
10  */
11
12  public class Until {
13
14    public static void main(String argv[]) {
15      Integer i = new Integer(argv[0]);
```

<table>
<tr><td>Programmablauf-
plan . . ..pap</td><td>Struktogramm
. . ..nsd</td></tr>
<tr><td>While.pap
 
</td><td>While.nsd
 
</td></tr>
<tr><td>Entscheidungs-
tabelle . . ..et</td><td>Programmier-
sprache . . ..java</td></tr>
<tr><td>
<table>
<tr><td colspan="2">While.et</td><td>R1</td><td>R2</td></tr>
<tr><td>B1</td><td>p</td><td>J</td><td>N</td></tr>
<tr><td>A1</td><td>$\mathcal{A}$</td><td>X</td><td></td></tr>
<tr><td>A2</td><td>run While.et</td><td>X</td><td></td></tr>
<tr><td>A3</td><td>Ende</td><td></td><td>X</td></tr>
</table>
</td><td>While.java

<pre>
class While {
⋮
   while (p) {
      A;
   }
}
</pre>
</td></tr>
</table>

Legende:
↪Tabelle 3.2 auf Seite 148

Tabelle 3.5: *While*-Konstrukt mit dem Prädikat p und dem Konstrukt $\mathcal{A}$

```
16      int v = i.intValue();
17      System.out.println("Aufruf mit " +  v + ".");
18
19      do {
20          System.out.println("Run A");
21          v = v - 1;
22      }
23      while (!(v < 0));
24    }
25  }
26  // End of file: /u/bonin/mywww/ewi/widata/Until.java
27
```

Aufruf der Klasse `Until.class`

```
>java Until 3
Aufruf mit 3.
Run A
Run A
Run A
Run A
>java Until 0
Aufruf mit 0.
Run A
>
```

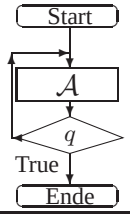
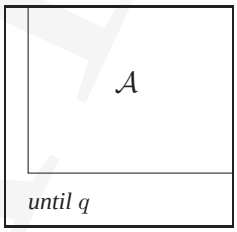
3.1.4 Nebenläufigkeit

Nebenläufigkeit (*concurrency*) liegt vor, wenn Prozesse (Aufgaben, Vorgänge etc.) voneinander unabhängig ablaufen (können). Für zwei nebenläufige Prozesse $\mathcal{C}_1$ und $\mathcal{C}_2$ ist es daher unerheblich, ob erst $\mathcal{C}_1$ und dann $\mathcal{C}_2$ oder ob erst $\mathcal{C}_2$ und dann $\mathcal{C}_1$, oder ob $\mathcal{C}_1$ und $\mathcal{C}_2$ gleichzeitig ablaufen. Daher ist Nebenläufigkeit ein umfassenderer Begriff als Parallelität, obwohl beide umgangssprachlich oft im Sinne von Synonymen verwendet werden.

Die ET kann eine Nebenläufigkeit nicht abbilden. In der Programmiersprache Java^{TM} läßt sich die Nebenläufigkeit durch *Threads* präzise abbilden[5] (↪Tabelle 3.7 auf Seite 158).

Thread

[5] An dieser Stelle sei nur auf die umfangreichen Quellen im Web hingewiesen. Ein kleines Beispiel zeigt `http://as.fhnon.de/JavaSAP/`.

Programmablauf-plan . . ..pap	Struktogramm . . ..nsd
Until.pap 	**Until.nsd** 
Entscheidungs-tabelle . . ..et	**Programmier-sprache . . ..java**

Until.et

		R1
A1	$\mathcal{A}$	X
A2	*run* Repeat.et	X

Repeat.et

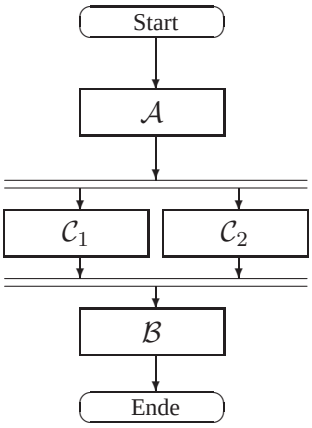
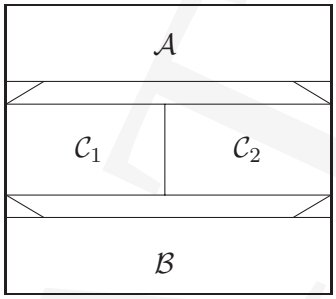
		R1	R2
B1	q	J	N
A1	$\mathcal{A}$		X
A2	*run* Repeat.et		X
A3	*return*	X	

Until.java

```
class Until {
⋮
   do {
      A
   }
   while (!q);
}
```

Legende:
↪Tabelle 3.2 auf Seite 148

Tabelle 3.6: *Until*-Konstrukt mit dem Prädikat q und der Funktion $\mathcal{A}$

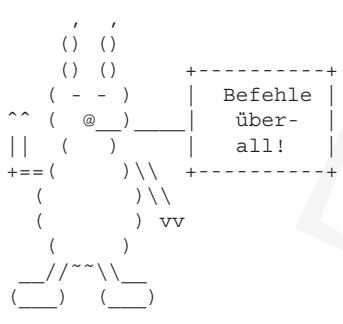
Programmablauf-plan . . ..pap	Struktogramm . . ..nsd
CC.pap  	**CC.nsd** 
Entscheidungs-tabelle . . ..et	**Programmier-sprache . . ..java**

Nicht vorgesehen!	umfassendes Thread-Konzept
	mit gegenseitiger
	Aktivierung und Deaktivierung
	CC.java
	`class CC {`
	⋮
	`new Thread(`$\mathcal{C}_1$`).start();`
	`new Thread(`$\mathcal{C}_2$`).start();`
	`}`

Legende:
↪Tabelle 3.2 auf Seite 148

Tabelle 3.7: Nebenläufigkeit (*Concurrency*) der Prozesse $\mathcal{C}_1$ und $\mathcal{C}_2$

3.2 Interpretation von Kommandos



Zu einem Betriebssystem gehört standardmäßig mindestens ein Kommandointerpreter ≡ *„Shell“*. Die Bezeichnung verdeutlicht, daß eine Shell[6] den Kern des Betriebssystems analog zu einer Muschel umschließt. Der Kern wird gegenüber den Benutzern abgeschirmt. Erst die Shell macht die Dienstleistungen des Kerns dem Benutzer verfügbar.

Beim „Einloggen“ in eine Plattform wird eine Shell[7] aufgerufen. Sie interpretiert die interaktiv eingegebenen oder in Dateien gespeicherten Kommandos. Beispielsweise ist es beim IBM UNIX-Betriebsystem AIX die Korn-Shell, kurz **ksh**.

Die Korn-Shell wurde von David G. Korn in den Bell-Laboratorien, einer weltweit hoch anerkannten Forschungsstätte von AT&T, konzipiert und für verschiedene Betriebssysteme entwickelt. Seit 1982 trat sie ausgehend von ihrem betriebsinternen Einsatz bei AT&T den „Siegeszug“[8] an. Die Korn-Shell ist eine leistungsfähige **höhere Programmiersprache**. In vielen Fälle kann mit ihr einfacher und wesentlich schneller als mit anderen Hochsprachen ein Programm entwickelt werden. Sie ist daher besonders geeignet für das sogenannte **Prototyping**.

ksh

[7] Bedeutsame Shells sind:

sh Bourne-Shell — entwickelt von Steven Bourne, Bell-Laboratorien (AT&T)

csh C-Shell — entwickelt von Bill Joy, University of California

ksh Korn-Shell — entwickelt von David G. Korn, Bell-Laboratorien (AT&T) ↪[Bo+91]

ruby Object-oriented Shell Ruby — entwickelt von Yukihiro Matsumoto („Matz“), Japan, `http://www.ruby-lang.org/en/` (online 28-Nov-2003) ↪ [Bo03]

Mit welcher Shell man gerade arbeitet, ist mit dem **ps**-Kommando feststellbar.

[8] Warum als Beispiel gerade ksh? Ksh ist umfangreicher als `csh` oder `bsh`, ein Beispiel ist das Exportieren von Funktionen. Trotz ihres größeren Umfanges bietet `ksh` eine bessere Performance, das heißt Programme werden schneller ausgeführt als mit einer der anderen Shells ([Bo+91] S. V). `ksh` kann alles was `csh` kann, jedoch mit `bsh`-kompatibler Syntax, zum Beispiel: *history*-Mechanismus, *aliasing*, *foreground* & *background* jobs, echtes Edieren der Kommandozeile (*emacs* & *vi* Modus).

Ein einfaches Kommando hat die folgende Form:

$ kommando [argumente]

Prompt

Dabei steht das Dollarzeichen „$“ für die Eingabeaufforderung (Prompt) der Shell, die eckigen Klammern für eine optionale Angabe[9] der Argumente.

3.2.1 Prozeß & *Pipe*

Erkennt die Shell die Angabe kommando als den Namen eines Kommandos, dann setzt sie (in der Regel) einen Kindprozeß ab, der versucht, das Kommando auszuführen. Der Kindprozeß selbst kann wiederum einen Kindprozeß generieren, wenn das Kommando aus primitiveren Kommandos zusammengesetzt ist (↪Bild 3.1 Seite 161). Ein Prozeß „kommuniziert“ mit dem Bildschirm (≡ Standardausgabe, Standardfehlerausgabe) und der Tastatur (Standardeingabe).

Umlenkung von Standardeingabe und Standardausgaben

Die Standardeingabe kann aus einer Datei erfolgen (eingabeDatei). Die Standardausgabe (ausgabeDatei) und Standardfehlerausgabe (fehlerDatei) können in Dateien umgelenkt werden, wenn diese Dateien mit den Richtungsanzeigern kleiner „<“ und größer „>“ angegeben werden; also wie folgt notiert werden:

$ kommando [argumente] <eingabeDatei >ausgabeDatei 2>fehlerDatei

Das Bild 3.2 auf der Seite 161 skizziert dieses Datenströme. Sind die angegebenen Ausgabedateien noch nicht vorhanden, dann werden sie angelegt. Ist eine geannte Ausgabedatei schon vorhanden, dann wird sie mit den neuen Daten überschrieben. Sollen die neuen Daten, die alten nicht überschreiben, sondern ergänzen, dann ist ein doppeltes Größerzeichen „>>„ als Richtungsanzeiger anzugeben. Dieses bewirkt ein Anhängen (*append*) der neuen Daten an die alten. Das `cat`-Kommando[10] gibt

[9] [. . .] ≡ Angabe kann entfallen.

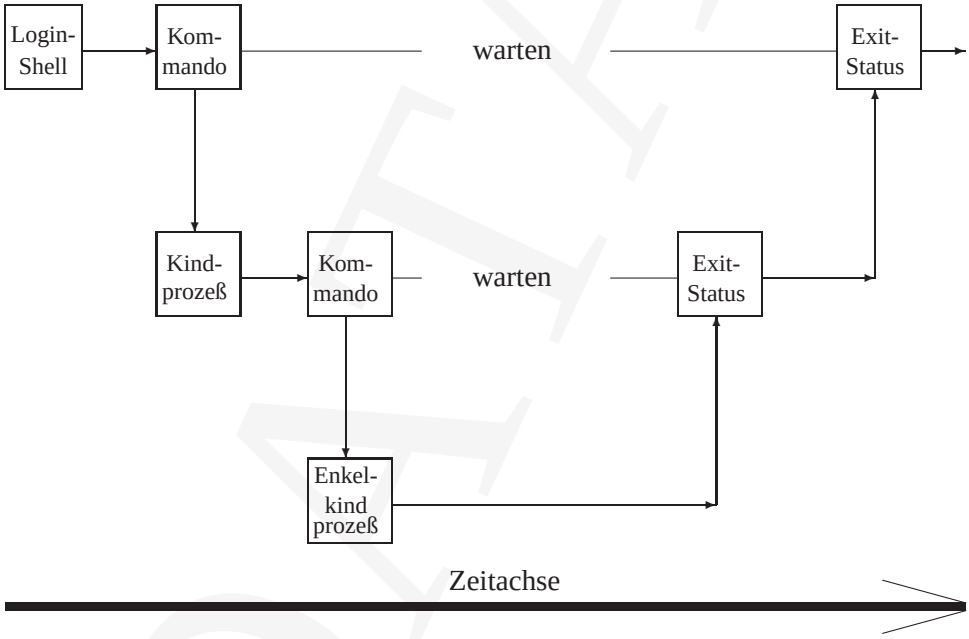


Abbildung 3.1: Abarbeitung eines Kommandos

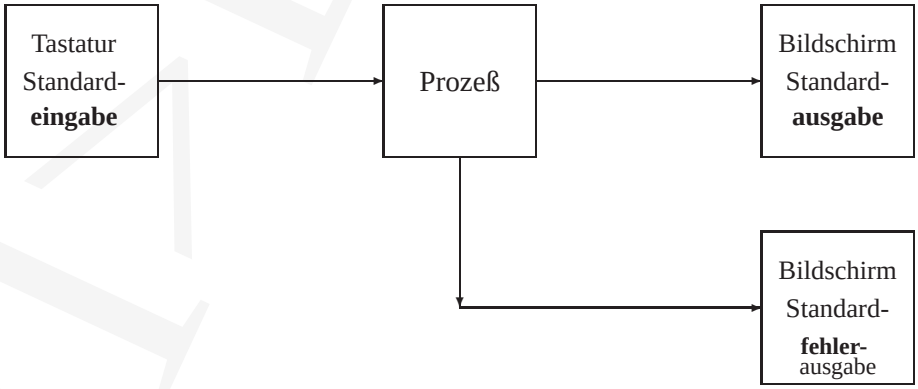


Abbildung 3.2: Datenstromskizze: Standardeingabe und Standardausgaben

den Inhalt einer Datei aus.

```
$ cat DV-Bedarf.tex >sicher/DV-Bedarf.tex 2>message
```

Hier versucht es, die Datei `DV-Bedarf.tex` zu lesen und den Inhalt in die Datei `sicher/DV-Bedarf.tex` auszugeben. Im Fehlerfall wird eine Nachricht in die Datei `message` geschrieben. Da die Standardausgabe und -fehlerausgabe umgelenkt sind, erscheint nach Ausführung nur das Promptzeichen ($) auf dem Bildschirm.

```
$ cat < message
⇒
cat: 0652-050 Cannot open DV-Bedarf.tex
$
```

Back Slash

Reicht eine Kommandozeile zur Eingabe nicht aus, dann ist ein umgekehrter Schrägstrich (*back slash*) „\ “ als Fortsetzungzeichen einzugeben. Die Shell zeigt dann einen anderen Prompt, bei `ksh` das Größerzeichen „>“.

```
$ cat DV-Bedarf.tex \
> >sicher/DV-Bedarf.tex 2>message
⇒
$
```

Pipe

Pipe

Mehrere Kommandos lassen sich zu einer *Pipe* verketten, das heißt, die Standardausgabe eines Kommandos ist gleich der Standardeingabe des nachfolgenden Kommandos.

$$\$\ \text{kommando}_1 \mid \text{kommando}_2 \mid \ldots \mid \text{kommando}_n$$

Die Standardausgabe der gesamten *Pipe* ist die Standardausgabe des zuletzt ausgeführten Kommandos. Die *Pipe* bildet einen Prozeß, der Kindprozesse zur Abarbeitung der einzelnen Kommandos absetzt. Das folgende Beispiel verknüpft die beiden Kommandos `ls` und `wc`:

`ls`

[10] *concatenate* ≡ verknüpfen, verketten

```
$ ls /u/bonin | wc -w
⇒
54
$
```

Das Kommando `ls` listet die Dateien im Verzeichnis (*directory*) `/u/bonin` auf. Das Ergebnis ist die Eingabe für das Kommando `wc` (word count). Mit dem Parameter `-w` zählt `wc` Wörter (mit -l Zeilen und mit -c Character). Am Bildschirm erscheint die Anzahl der Dateien in `/u/bonin`. WC

Foreground Job⇔Background Job

Der Login-Prozeß wartet normalerweise, bis ein Kommando vollständig ausgeführt ist, ehe wieder der Prompt erscheint. In dieser Ausführungszeit kann man keine anderen Aufgaben bearbeiten. Will man vor der vollständigen Abarbeitung jedoch weitermachen, dann ist das Kommado als Hintergrundprozeß zu starten.

Einen Hintergrundprozeß startet man, indem man am Ende der Eingabezeile[11] das Ampersandzeichen „&“ eingibt. Jeder Hintergrundprozeß erhält eine eigene PID (*Process Identification Number*). Ohne Voreinstellung gehen die Standardausgabe und Standardfehlerausgabe auf den Bildschirm, während die Standardeingabe die eine vorgegebene Datei[12] ist, damit die Eingaben von der Tastatur zum Weiterarbeiten genutzt werden können. Im folgenden Beispiel wird das Kommando `mykommando` als Hintergrundprozeß gestartet. Sein Ergebnis steht in der Datei `ergebnis`.

PID

```
$ mykommando > ergebnis &  # Ausführung im Hintergrund
⇒
[1]  5236 # PID=5236, JobID=1

$ fg  %1 # Holt Job mit Nummer 1 wieder in den Vordergrund.

$ <CRTL-z> # Stoppt den Vordergrundjob

$ bg # Stellt den Vordergrundjob wieder in den Hintergrund.
```

[11] Steht & in der Mitte einer Kommandozeile, dann hat es dieselbe Wirkung wie eine Entertaste, also wie ↩.

[12] Zum Beispiel `/dev/null`

Wenn mehrere Jobs im Hintergrund laufen, kann man sich mit dem Kommando `jobs` einen Überblick verschaffen. Prozesse können mit dem Kommando `kill` vorzeitig abgebrochen werden. Dazu benötigt man die PID oder die JobID (Identifikation).

Daemon **&** *Zombie Process*

Daemon

Als *Daemon* bezeichnet man einen Prozeß, der üblicherweise beim Systemstart mit gestartet werden und bis zum Systemstop läuft. Ein *Daemon* stellt Systemleistungen bereit und ist im Regelfall für mehr als eine *Task* oder einen *User* verfügbar. Ein *Zombie Process* ist ein gestoppter Prozeß, der in der Prozeßtabelle existiert (≡ hat eine PID) dem aber keine anderer *System Space* zugeordnet ist. *Zombie Processes* existieren häufig solange bis der *Parent Process* endet.

Zombie Process

3.2.2 Input Interface

Eine Datei, die Kommandos enthält, wird als *Shell Script* oder hier kurz und eingedeutscht als Script bezeichnet. Die folgende Datei `gls` (german ls) ist ein einfaches Script-Beispiel. Es nutzt das `print`-Kommando[13], das eine Zeichenkette (*string*) auf dem Bildschirm ausgibt. Das Kommando `ls` listet die Dateien des aktuellen bzw. angegebenen Verzeichnisses auf. Der Parameter „`l`“ steht für die Langformausgabe.

```
 1  #!/bin/ksh
 2  #
 3  # @(#) gls V1.0 ist ein einfaches ls-Kommando
 4  # @(#) Aufruf: gls file or directory
 5  # @(#) Bonin 18-Nov-1999
 6  #
 7  ls -ld $1 | read Rechte Links Besitzer \
 8                   Gruppe Groesse Monat Tag Zeit Name
 9
10  print " ----------------\
```

[13] David G. Korn empfiehlt statt dem `echo`-Kommando das `print`-Kommando zu nutzen, da die Kompatibilität zwischen Bourne-Shell und Korn-Shell hier nicht immer gegeben ist ([Bo+91] S. 60). [Hinweis: Das erste Zeichen im String des `print`-Kommandos sollte kein „-“-Zeichen sein, weil sonst die Zeile nicht ausgegeben wird — bedeutsam beim Abbilden eines waagerechten Striches.]

```
11  ----------------------------------------"
12  print " Datei           : $Name"
13  print " Rechte          : $Rechte"
14  print " Groesse         : $Groesse Byte"
15  print " Besitzer        : $Besitzer"
16  print " Gruppe          : $Gruppe"
17  print " Letzte Aenderung: $Tag.$Monat"
18  print " ----------------\
19  ----------------------------------------"
20  # End of object: gls
21
```

Rechte setzen

Damit die Datei `gls` für jederman ausführbar ist, setzen wir für die drei Klassen:

`u` `user` (*owner*),

`g` `group` und

`o` `others`

das *execute*-Recht mit Hilfe des `chmod`-Kommandos. Mit dem Argument `a` (für *all*, also für `u`,`g` und `o`) wird zusätzlich (Argument `+`) das sogenannte `x`-Recht gegeben: `chmod`

```
$ chmod a+x gls
⇒
$
```

Damit keiner die (mühsam) erstellte Datei `gls` (versehentlich) ändert, wird der Datei das Schreibrecht (`write`) entzogen.[14]

```
$ chmod a-w gls
⇒
$
```

Nun läßt sich das Script `gls` mit einem Argument, zum Beispiel dem Pfad- und Dateinamen `edv/edv-all.dvi`, aufrufen. Die erste `gls`-Zeile beginnt mit `#!` – dem Kommentarzeichen gefolgt vom Ausrufezeichen. Anschließend folgt der Pfadname der Shell, unter der die folgenden Zeilen abgearbeitet werden sollen. Mit der Angabe `/bin/ksh` in der ersten Zeile wird erzwungen, daß `gls` von der Korn-Shell ausgeführt wird, selbst wenn `gls` von einer anderen Shell aus, zum Beispiel der C-Shell, gestartet wird. Mit der Zeichenfolge `@(#)` in den folgenden Zeilen ist der eine Kurzbeschreibung notiert, die als sogenannter `#!` `@(#)`

[14] Das Leserecht hat den Kennbuchstaben „`r`“(`read`).

what string bezeichnet wird, weil mit Hilfe des `what`-Kommandos diese Kommentarzeilen direkt anzeigbar sind.

```
$ what gls
⇒
gls:
gls V1.0 ist ein einfaches ls-Kommando
Aufruf: gls file or directory
Bonin 18-Nov-199
$
$ gls edv/edv-all.dvi
⇒

-----------------------------------------------------------
Datei           : edv/edv-all.dvi
Rechte          : -rw-r-----
Groesse         : 225892 Byte
Besitzer        : bonin
Gruppe          : staff
Letzte Aenderung: 18.Nov
-----------------------------------------------------------

$
```

Das Script `gls` erhält seine Eingaben zum einen in Form der Parameterbindung `$1` und zum anderen durch das `ls`-Kommando und die *pipe* ins `read`-Kommando. In ein Script können Daten auf die folgenden drei Arten eingebracht werden:

Parameter Durch Bindung der Parameter mittels Angabe von Argumenten beim Script-Aufruf.

Datei Durch das Lesen einer Datei, beispielsweis mit dem `read`-Kommando; wobei das Lesen von der Standardeingabe (Tastatur) oder ihrer „Umlenkung“ erfolgt.

Environment Über die Kommunikation mit der Script-Ausführungsumgebung (≡ *Environment*), das heißt Nutzung der vorher gesetzten Werte von Umgebungsvariablen.

Parameter

`$n`

Im Shell-Script können die Bindungen der vordefinierten Variablen `$1, $2, ...,$9, ${10} ,..., ${n}` genutzen werden. Beim Aufruf

des Scripts wird an den Parameter `$1` der Wert des ersten Argumentes gebunden – das zweite Argument an `$2`, das dritte an `$3` und so weiter. Bis zum 10. Argument besteht der Parametername aus einem Dollarzeichen direkt gefolgt von der entsprechenden Zahl. Vom 10. Argument an ist die Zahl in geschweiften Klammern anzugeben. Die mögliche Zahl von Argumenten wird durch den Gesamtspeicherbedarf aller Argumente bestimmt. Da die Parameter in der Reihenfolge der Argumente ihre Werte bekommen, spricht man von **Stellungsparametern** oder auch **Positionsparametern**. Das folgende Script-Beispiel `param.in` zeigt die Bindungen dieser vordefinierten Dollar-Variablen:

```
 1  #!/bin/ksh
 2  #
 3  # @(#) param.in V1.0 verdeutlicht die Parameterbindung
 4  #                    und der vordefinierten Variablen
 5  #
 6  #
 7  find / -user guest -print 2>/dev/null & # Hintergrundprozess
 8  #
 9  print " ---------------------------------------\
10  --------------------"
11  print " Script                                 : $0"
12  print " 1. Parameter                           : $1"
13  print " 2. Parameter                           : $2"
14  print " 3. Parameter                           : $3"
15  print " DollarStern -- ein String              : $*"
16  print "                String-Laenge           :\
17   $(print  $* | wc -c)"
18  print " DollarKlammeraffe -- einzelne Zeichenk.: $@"
19  print " DollarHash -- Anzahl der Argumente     : $#"
20  print " DollarDollar -- PID aktuelle Shell     : $$"
21  print " DollarAusrufezeichen -- PID Background : $!"
22  print " DollarFragezeichen -- ExitSatus        : $?"
23  print "                0 = erfolgreich           "
24  print "                1 = nicht erfolgreich     "
25  print " ---------------------------------------\
26  --------------------"
27
28
```

In diesem Script[15] wird ein Hintergrundprozeß gestartet, um den

[15] [Hinweis: Das `wc`-Kommando stellt bei einem leeren String die Länge = 1 fest:
`$ print "" | wc -c`

Wert der `$!`-Variablen zeigen zu können. Dieser Hintergrundprozeß durchsucht aufgrund des `find`-Kommandos alle Dateien ausgehend von der Wurzel des Dateiensystems `/` nach dem User `guest` und leitet auftretende Fehlermeldungen in den Papierkorb `/dev/null` um.

```
$ param.in eins zwei drei vier
⇒
 ------------------------------------------------------------
 Script                                  : ./param.in
 1. Parameter                            : eins
 2. Parameter                            : zwei
 3. Parameter                            : drei
 DollarStern -- ein String               : eins zwei drei vier
                String-Laenge            : 20
 DollarKlammeraffe -- einzelne Zeichenk.: eins zwei drei vier
 DollarHash -- Anzahl der Argumente      : 4
 DollarDollar -- PID aktuelle Shell      : 11536
 DollarAusrufezeichen -- PID Background : 12561
 DollarFragezeichen -- ExitSatus         : 0
                0 = erfolgreich
                1 = nicht erfolgreich
 ------------------------------------------------------------
$ /usr/lpp/bos/inst_root/home/guest
/var/spool/mail/guest
/home/guest
/home/guest/readme
/home/guest/.emacs
/home/guest/.profile
/home/guest/TeachText
/home/guest/mbox
/home/guest/.aic.errlog
/home/guest/.sh_history
$
```

Environment

Die Script-Abarbeitung ist abhängig von der Ausführungsumgebung (*Environment*). Von jeder *Environment*-Variablen, die mit Hilfe des `export`-Kommandos allgemein verfügbar gemacht wird, erhält das Script beim

---

⇒ 1
]

Aufruf eine Kopie.

Daten, die sich nur selten ändern, brauchen nicht bei jedem erneuten Aufruf als Argument übergeben zu werden. Sie lassen sich einfacher als exportierte *Environment*-Variable in das Script transferieren. Das folgende Script `kasse.in` zeigt ein entsprechendes Beispiel. Dabei wird das `let`-Kommando für elementare Integeroperationen (Addieren, Subtrahieren, Multiplizieren, Dividieren) genutzt. Mit der `integer`-Deklaration[16] der anwendungsspezifischen Variablen, wie zum Beispiel `UmsatzSteuerSatz`, wird eine Typprüfung durch die Korn-Shell erreicht. Mit der Kontrollstruktur `if ... then ... else ... fi` wird sichergestellt, daß die Werteberechnung nur bei einem Aufruf mit genau zwei Argumenten erfolgt.


```
 1  #!/bin/ksh
 2  #
 3  # @(#) kasse.in V1.0 berechnet die Kassenforderung
 4  #                    in ganzen DM
 5  #
 6  # Aufruf: kasse.in Skonto Nettopreis
 7  #
 8  # Environment-Variable: MWST
 9  # Achtung: Integer-Rundung!
10  #
11  if [ "$#" -ne 2 ]          # Anzahl der Argumente pruefen
12  then
13    print "Aufruf: $0 Skonto Nettopreis"
14    exit 1                   # Exit-Status fuer $? setzen
15  else
16    integer UmsatzSteuerSatz UmsatzSteuerBetrag \
17            Brutto Forderung Rabatt
18    let "UmsatzSteuerSatz    = $MWST"
19    let "UmsatzSteuerBetrag  = ($2 * UmsatzSteuerSatz) / 100"
20    let "Brutto              = $2 + UmsatzSteuerBetrag"
21    let "Rabatt              = (Brutto * $1) / 100"
22    let "Forderung           = Brutto-Rabatt"
23    #
24    print " --------------------\
25  ------------------------------------"
26    print " Nettopreis          : $2 DM"
27    print " Umsatzsteuer-Satz   : $UmsatzSteuerSatz Prozent"
```

[16] Die Korn-Shell kennt die Datentypen: Zeichenkette (`String` = *Default*-Wert), Ganzzahlig (`integer`) und Feld (`array`).

```
28    print " Umsatzsteuer-Betrag : $UmsatzSteuerBetrag DM"
29    print " Bruttopreis         : $Brutto DM"
30    print " Skonto              : $1 Prozent"
31    print " Rabatt              : $Rabatt DM"
32    print " Forderung           : $Forderung DM"
33    print " --------------------\
34  ---------------------------------------"
35    exit 0
36  fi
37
```

Vor dem Script-Aufruf wird die benötigte *Environment*-Variable `MWST` zugewiesen und exportiert.

```
$ MWST=16 # Mehrwertsteuersatz 1999
⇒ $
$ export MWST
⇒
$
```

Das Script `kasse.in` kann jetzt mit den beiden Argumenten für Skonto und Nettopreis aufgerufen werden. Der Mehrwertsteuersatz wird über die *environment*-Variable `MWST` bereitgestellt. Ein drittes Argument wird eingespart. Dieser „Seiteneinstieg“ birgt jedoch Überschaubarkeitsprobleme – er ist daher nicht ganz ungefährlich.

```
$ kasse.in 2 300
⇒
 ------------------------------------------------------------
 Nettopreis          : 300 DM
 Umsatzsteuer-Satz   : 15 Prozent
 Umsatzsteuer-Betrag : 45 DM
 Bruttopreis         : 345 DM
 Skonto              : 2 Prozent
 Rabatt              : 6 DM
 Forderung           : 339 DM
 ------------------------------------------------------------
$
```

Die Stellungsparameter müssen natürlich weiterhin in der richtigen Reihenfolge „bedient“ werden.

```
$ kasse.in 300 2
⇒

 ------------------------------------------------------------
 Nettopreis          : 2 DM
 Umsatzsteuer-Satz   : 15 Prozent
```

```
 Umsatzsteuer-Betrag : 0 DM
 Bruttopreis         : 2 DM
 Skonto              : 300 Prozent
 Rabatt              : 6 DM
 Forderung           : -4 DM
 ------------------------------------------------------------
$
```

Die Typen der Argumente müssen für die Integerarithmetik passen, andernfalls meldet die Korn-Shell einen Fehler. Um die weitere Abarbeitung zu ermöglichen, wird jedoch dann der typentsprechende Ersatzwert genutzt.

```
$ kasse.in bonin 1000
⇒

./kasse.in[14]: Rabatt=(Brutto*bonin)/100: 0403-009
        The specified number is not valid for this command.
------------------------------------------------------------
Nettopreis          : 1000 DM
Umsatzsteuer-Satz   : 15 Prozent
Umsatzsteuer-Betrag : 150 DM
Bruttopreis         : 1150 DM
Skonto              : bonin Prozent
Rabatt              : 0 DM
Forderung           : 1150 DM
------------------------------------------------------------
$
```

Auch mit einem Argument vom falschen Typ wird das Script im `else`-Zweig mit dem Komanndo `exit 0` beendet. Der Wert von `$?` ist daher gleich Null.

```
$ print $?
⇒
0
$
```

Wird nur ein Argument angegeben, dann ist der Exit-Stutus gleich 1.

```
$ kasse.in 1000
⇒
Aufruf: ./kasse.in Skonto Nettopreis
```

```
$ print $?
⇒
1
$
```

Man beachte stets die Groß/Kleinschreibung bei der Korn-Shell.

```
$ Kasse.in 2 1000
⇒
/bin/ksh: Kasse.in: not found.
$
```

3.3 Übungen

3.3.1 Kontrollstruktur analysieren

Die Bild 3.3 auf Seite 173 zeigt die Funktion $FOO(x, y, z)$ als Struktogramm (Nassi/Shneiderman-Diagramm).

Funktionswert bestimmen

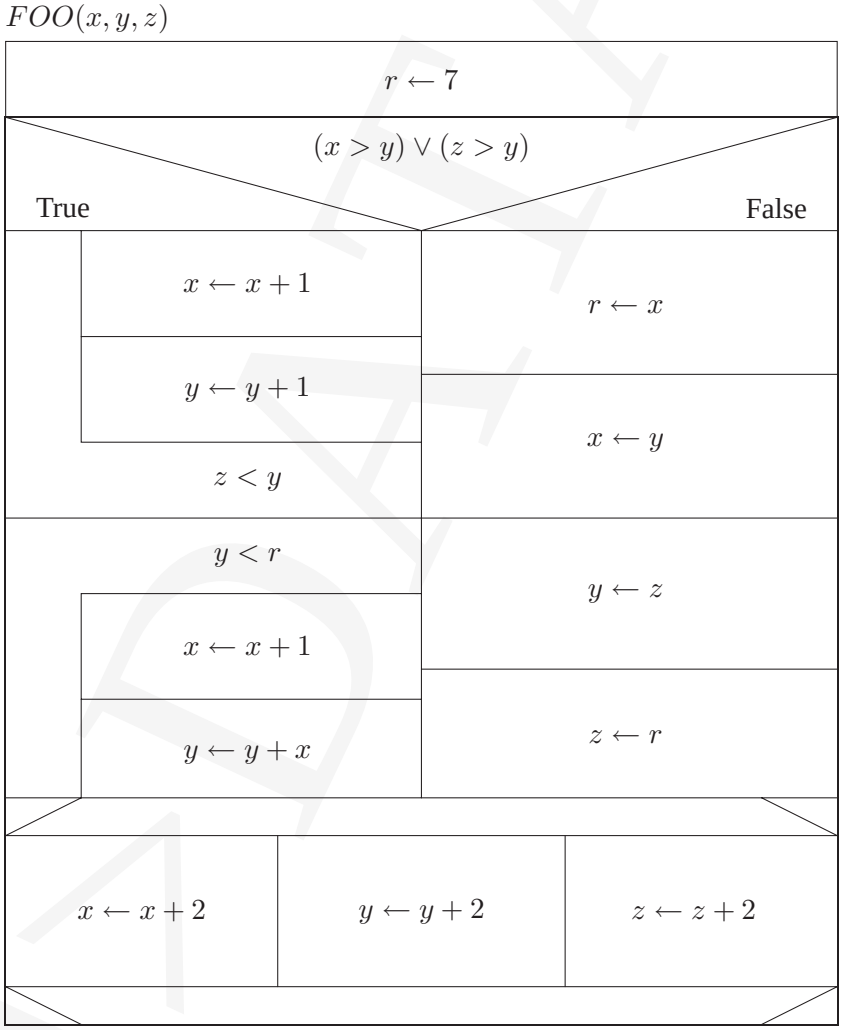
Es wird $FOO(1, 2, 3)$ aufgerufen. Es gibt daher folgende Bindungen:

- $x \leftarrow 1$
- $y \leftarrow 2$
- $z \leftarrow 3$

Geben Sie die Werte für x, y, z und r nach der FOO-Ausführung an.

Elementare Konstrukte erkennen

Aus welchen elementaren Konstrukten besteht die Funktion $FOO(x, y, z)$? Nennen Sie diese.



Legende:
$\mathcal{A} \leftarrow \mathcal{B} \equiv$ Zuweisungsbefehl: $\mathcal{A}$ erhält den Wert von $\mathcal{B}$

Abbildung 3.3: Struktogramm der Funktion $FOO(x, y, z)$

Struktur in XML notieren

Skizzieren Sie die FOO-Struktur in XML. Abstrahieren Sie dabei von den einzelnen Zuweisungen und Prädikaten. Nehmen Sie dafür einfach das Element `<STATEMENT />` an.

3.3.2 *Pipe*-Konstrukt interpretieren

Die Leistungen eines Betriebssystems werden über ein Dienstprogramm, die sogenannte *Shell*, für den Benutzer direkt verfügbar. In der Unix-Welt ist die Korn-Shell (ksh) weit verbreitet. Frau Emma Musterfrau, eine erfahrene ksh-Programmiererin, hat für ein kleines Alltagsproblem das folgende ksh-Skript `quartal.ksh` geschrieben:

`quartal.ksh`

```
 1  #!/bin/ksh
 2  #
 3  # Autor:  Emma Musterfrau
 4  # Version: 1.0 01-Apr-99
 5  #
 6  # @(#) quartal.ksh sammelt Verkaufspositionen
 7  # @(#) Syntax: quartal.ksh
 8  #
 9  # Ouartalsdatei schreiben
10  #
11  rm Quartal0199.txt
12  cat < VerkaufJan.txt >  Quartal0199.txt
13  cat < VerkaufFeb.txt >> Quartal0199.txt
14  cat < VerkaufMar.txt >> Quartal0199.txt
15  #
16  # Anzahl der Verkaufspositionen
17  #
18  typeset -i anzahl=`cat < Quartal0199.txt | wc -w`
19  print "Quartal0199.txt wurde erzeugt!"
20  print "Anzahl der Positionen: $anzahl"
21  #
22  # Quelle:
23  # 193.174.33.106:/u/bonin/mywww/ewi/ss99/Verkauf/quartal.ksh
24
```

Frau Emma Musterfrau zeigt sich den Inhalt der folgenden Dateien mit Hilfe des Kommandos `cat` an. [Hinweis: Als Promptzeichen wird hier das Dollarzeichen („$“) verwendet.]/

```
$cat < VerkaufJan.txt
100.00
200.00
300.00
$cat < VerkaufFeb.txt
800.00
700.00
$cat < VerkaufMar.txt
100.00
400.00
600.00
500.00
```

Inhalt der Datei `Quartal0199.txt`

Geben Sie genau den Inhalt der Datei `Quartal0199.txt` an, wenn das Skript `quartal.ksh` einmal ausgeführt wurde; also die Anzeige von:
`$cat < Quartal0199.txt`

Ausgabe auf dem Bildschirm

Wenn das Skript `quartal.ksh` ausgeführt wird, dann entsteht eine Ausgabe auf dem Bildschirm. Geben Sie diese Bildschirmausgabe exakt an.

`what`**-Kommando**

Frau Emma Musterfrau hat auf ihrem Rechner mehr als 1000 ksh-Programme. Manchmal weiß sie nicht mehr was ein Skript leistet. Sie nutzt dann das `what`-Kommando. Geben Sie das Ergebnis für das Skript `quartal.ksh` an:

```
$what quartal.ksh
```

3.3.3 Sichere Kommunikation im Netz

Das Beratungsunternehmen *LISTIG* GmbH (Liebermann & Steinhauer & Ignorant) beschäftigt weltweit ca. 250 Berater. Die Kommunikation zwischen den Berater untereinander und mit der Geschäftsleitung erfolgt vorzugsweise mit elektronischer Post (*email*) über das Internet. In der letzten Zeit häufen sich die Fälle von Verfälschung von Nachrichten und von Vortäuschung falscher Absender. In beiden Fällen entstehen der *LISTIG* GmbH dadurch erhebliche Schäden.

Vorschlag für sichere *Email* skizzieren

Skizzieren Sie einen Lösungsvorschlag, damit solche Schäden in Zukunft nicht mehr vorkommen können.

Aufgaben eines *Key Servers* erläutern

Bei der Diskussion in der Geschäftsleitung erwähnt der DV-Experte Hans Weisheit den Begriff „*Key Server*“. Beschreiben Sie kurz welche Schlüssel ein solcher Server verwaltet.

***Email* oder `ftp`-basierte Kommunikation?**

Im Rahmen der Diskussion schlägt Herr Fritz Schlaumeier vor, die Kommunikation auf `ftp` (*file transfer protocol*) umzustellen. Was halten Sie von diesem Vorschlag? Skizzieren Sie Ihre Kritik.

3.3.4 Client⇔Server-Architektur

Das DV-Equipment beim Autoersatzteilehersteller *Mechanik-Lüneburg AG* (*ML-AG*) mit einem Umsatz von 400 Mio Euro und 150 Beschäftigten ist in den letzten drei Jahrzehnten ohne große Planung gewachsen. Viele Daten werden daher mehrfach in unterschiedlichen DV-Systemen gespeichert.

Beispielsweise verwaltet die Einkaufsabteilung die Adressen ihrer Geschäftspartner mit dem selbst programmierten COBOL-System *E-SYS* (*Einkaufssystem*), welches auf einem IBM-Computer vom Typ AS/400 läuft. An diesen Computer sind zur Zeit 25 Terminals (PCs) angeschlossen. Die Marketingabteilung speichert ihre Geschäftspartner mit dem 11-Jahre alten System *M-Access* (*Marketing-Access*) des kleinen Softwarehauses *Innovativ GmbH*. Dieses System läuft lokal jeweils auf den 5 PCs der Marketingabteilung. Diese PCs sind über ein LAN miteinander und mit dem Computer AS/400 verknüpft. Nur an einem der 5 PCs können die Daten gepflegt werden. Für die anderen 4 PCs werden die Daten jeden Abend durch File-Transfer aktualisiert.

Häufig sind dieselben Geschäftspartner in beiden Systemen gespeichert. Nicht selten mit leicht abweichenden Firmenkurzbezeichnungen und Schreibweisen der Adressen. Es kommt vor, daß eine geänderte Adresse eines Geschäftspartners nur in *E-SYS* und nicht auch in *M-Access* angepaßt wurde, oder umgekehrt.

Die Geschäftsführung hat beschlossen moderne Hard- und Software zu beschaffen und die neue DV-Infrastruktur konsequent als eine *Client-Server*-Architektur zu realisieren. Die veralteten PCs und die AS/400 sollen möglichst bald außer Betrieb genommen werden.

Client⇔Server-Lösung skizzieren

Was versteht man in diesem Zusammenhang unter einer *Client-Server*-Architektur? Skizzieren Sie eine mögliche *Client-Server*-Lösung für die Verwaltung der Adressen der Geschäftspartner.

Web-Browser nutzen

Zur Vereinfachung der Handhabung soll für jede Applikation die Benutzungsoberfläche über einen Web-Browser realisiert werden. Skizzieren Sie die *Server* und deren Verknüpfungen, damit die Ein- und Ausgaben einer Applikation beim *Client* über einen marktüblichen Web-Browser erfolgen können.

3.3.5 Kontrollstruktur präzise notieren

Eine nicht abweisende Schleife („fußgesteuerte“ Schleife) soll solange durchlaufen wie die Bedingung q nicht erfüllt ist. In der Schleife werden die Blöcke A, B, C und D nacheinander ausgeführt. Allerdings wird vor einer Ausführung des letzten Blockes D die Bedingung p geprüft. Ist $p = true$ wird D ausgeführt. Ist $p = false$ wird D nicht ausgeführt. Da es sich hierbei um einen Ausschnitt eines größen Programm handelt, dürfen p und q nicht verändert werden, da sie später gegebenenfalls doch noch benötigt werden.

Struktogramm-Notation

Notieren Sie diese textliche Ablaufbeschreibung in Form eines Struktogrammes.

Java^{TM}-Notation

Notieren Sie Ihr Struktogramm als Java^{TM}-Applikation.

3.3.6 ET-Verbundsystem aufstellen

Ein Dienstleistungsunternehmen plant eine Werbeaktion. Es ist daher beabsichtigt, die Eintrittspreise durch eine Rabattgewährung attraktiver zu gestalten. Der Marketingleiter formuliert als ersten Entwurf folgenden Werbetext: „Es wird ein Rabatt von 5% gewährt, wenn die Karte 30 Werktage vor der Veranstaltung gekauft wird. Für eine Gruppe von 7 oder mehr Personen, die so frühzeitig ihre Karten kauft, wird sogar 10% Rabatt gewährt. Eine Gruppe erhält auch dann einen Rabatt, wenn sie später kauft. Allerdings müssen die Karten dann 3 Werktage vorher gekauft werden. In diesem Fall beträgt der Rabatt jedoch nur 8%. Da Jugendliche, das heißt Personen jünger als 14 Jahre, schon heute nur 50% des normalen Preises zahlen müssen, gilt die Rabattregelung dieser Werbeaktion für sie nicht.“

Stellen Sie diese Werbeaktion mit Hilfe von mehreren verknüpften Entscheidungstabellen (ET-Verbundsystem) dar.

3.3.7 ET-Verbundsystem `Buchung` analysieren

Die Buchhalterin Emma Klarmeyer notiert die drei Entscheidungstabellen **ET-Buchung**, **ET-Saldierung** und **ET-Nachforschung** (↪Tabelle 3.8) um das Buchungsgeschehen in Ihrem Betrieb, der *Brillianz GmbH, Lüneburg*, zu präzisieren.

Entscheidungstabellen klassifizieren

Um welche Art von Entscheidungstabellen handelt es sich? Geben Sie den genauen Typ von jeder einzelnen Entscheidungstabelle an.

Formale Vollständigkeit feststellen

Stellen Sie fest, ob alle formal möglichen Fälle in jeder ET abgebildet sind. Wenn nicht, dann nennen Sie die fehlende(n) Bedingungsanzeigerfolge(n).

ET konsolidieren

Prüfen Sie, welche von den obigen Entscheidungstabellen konsolidierbar sind. Konsolidieren Sie die entsprechenden Entscheidungstabelle(n) und beschreiben Sie diese in der konsolidierten Form. Hat die Konsolidierung Einfluß auf die Zahl der Bedingungen?

3.3.8 ET-Verbundsystem `Workflow` analysieren

Die Abteilungsleiterin Emma Maus notiert die Entscheidungstabellen **ET-Workflow**, **ET-Bearbeitung** und **ET-Überprüfung** (↪Tabelle 3.9) um das Geschehen in Ihrer Abteilung K (*Konstruktion*) der ReTAG (*Regelungstechnik AG*, München) zu präzisieren.

ET klassifizieren

Um welche Art von Entscheidungstabellen handelt es sich? Geben Sie den genauen Typ von jeder einzelnen Entscheidungstabelle an.

	ET-Buchung	R1	R2	R3	R4
B1	Belegnummer angegeben?	J	J	N	N
B2	Buchungszweck angegeben?	J	N	J	N
A1	$i \leftarrow 1$	X	X		
A2	$summe \leftarrow 0$	X	X		
A3	**run** ET-Saldierung	X	X		
A4	$summe$ auf Beleg notieren	X	X		
A5	An zuständigen Sachbearbeiter abgeben			X	
A5	**run** ET-Nachforschung				X

	ET-Saldierung	R1	R2	R3	R4
B1	$i \leq anzahlPositionen$	J	J	J	N
B2	$position[i]$ numerisch?	J	J	N	-
B3	$position[i] \geq 0$ Euro	J	N	-	-
A1	$summe \leftarrow summe + position[i]$	X	X		
A2	$i \leftarrow i + 1$	X	X	X	
A3	**run** ET-Nachforschung			X	
A4	**run** ET-Saldierung	X	X	X	
A5	return				X

	ET-Nachforschung	R1	R2
B1	Belegersteller bekannt?	J	N
A1	Zurück an Belegersteller	X	
A2	Vermerk auf Konto *Dubios*		X
A3	return	X	X

Tabelle 3.8: Buchungsgeschehen in der *Brillianz GmbH, Lüneburg*

ET-Workflow		R1	R2	R3	R4	R5	R6	R7	R8
B1	$\mathcal{A}$ hinreichend spezifiziert?	J	J	J	J	N	N	N	N
B2	Geeigneter $\mathcal{B}$ vorhanden?	J	J	N	N	J	J	N	N
B3	Ausreichend $\mathcal{Z}$ kalkuliert?	J	N	J	N	J	N	J	N
A1	Abstimmung mit Auftraggeber		X			X		X	
A2	$\mathcal{A}$ zurückgeben			X	X		X		X
A3	**run** ET-Bearbeitung	X							
A4	**run** ET-Workflow		X		X	X		X	

ET-Bearbeitung		R1	R2
B1	$\mathcal{B}$ arbeitet planmäßig?	J	N
B2	$\mathcal{A}$ fertig?	–	N
A1	$\mathcal{B}$ weiter unterstützen?	X	
A2	**run** ET-Überprüfung		X
A3	**run** ET-Bearbeitung	X	X
A4	return		

ET-Überprüfung		R1	R2
B1	$\mathcal{B}$ überfordert?	J	N
A1	$\mathcal{B}$ schulen & unterstützen	X	
A2	$\mathcal{Z}$ neu festlegen	X	X
A3	return	X	X

Legende:
$\mathcal{A} \equiv$ Aufgabe, Arbeitseinheit
$\mathcal{B} \equiv$ Sachbearbeiter
$\mathcal{Z} \equiv$ Bearbeitungszeit

Tabelle 3.9: Workflow in der K-Abteilung der ReTAG, München

Formale Vollständigkeit feststellen

Stellen Sie fest, ob alle formal möglichen Fälle in jeder ET abgebildet sind. Wenn ja, begünden Sie Ihre Aussage; wenn nicht, dann nennen Sie die fehlende(n) Bedingungsanzeigerfolge(n) und geben dazu sinnvolle Aktionsanzeigerfolge(n) an.

ET konsolidieren

Prüfen Sie, welche von den obigen Entscheidungstabellen konsolidierbar sind. Konsolidieren Sie die entsprechenden Entscheidungstabelle(n) und beschreiben Sie diese in der konsolidierten Form.

3.3.9 ET-Eigenschaft

Kann die Konsolidierung einer Eintreffer-ET prinzipiell Einfluss auf die Zahl ihrer Bedingungen haben? Begründen Sie kurz Ihre Aussage.

3.3.10 `String`-Manipulation & Sortierung

Herr Herbert Kruse, ein erfahrener *Korn-Shell*-Programmierer, hat für ein kleines Alltagsproblem das folgende ksh-Skript `work.ksh` geschrieben:

`work.ksh`

```
 1  #!/bin/ksh
 2  #
 3  # Autor:  Herbert Kruse
 4  # Version: 1.0, 31-Jan-2002
 5  #
 6  # @(#) work.ksh bereitet Textdaten auf
 7  # @(#) Syntax: work.ksh originalDatei aufbereiteteDatei
 8  #
 9  # Zeichenkonvertierung & Sortierung
10  #
11  tr "[A-Z]" "[a-z]" < $1 | sort > $2
12  #
13  # Anzahl der Datensaetze
```

```
14  #
15  typeset -i insgesamt=`cat < $2 | wc -l`
16  print "Die Datei $2 hat $insgesamt Zeilen."
17  #
18  # End of file: work.ksh
19
```

Herr Herbert Kruse zeigt sich den Inhalt der Datei `konstrukte.txt` mit Hilfe des Kommandos `cat` an. [Hinweise: Als Promptzeichen wird hier das Dollarzeichen („$“) verwendet. Das Zeichen „↩“ steht für das Drücken der Enter-Taste. Leerzeilen stehen nicht in der Datei.]

```
$cat < konstrukte.txt↩
<H3>
<P>
<HTML>
<Body>
<TITLE>
<OL>
<uL>
<H1>
<H2>
```

Eine Datei `ergebnis.txt` gibt es zunächst nicht.

Inhalt der Datei `ergebnis.txt`

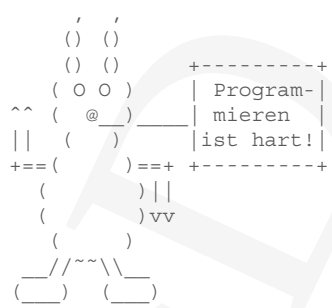
Geben Sie exakt den Inhalt der Datei `ergebnis.txt` an, wenn das Skript `work.ksh` wie folgt angewendet wurde:
`$work.ksh konstrukte.txt ergebnis.txt↩`

Ausgabe auf dem Bildschirm

Wenn das Skript `work.ksh` so angewendet wird, dann entsteht eine Ausgabe auf dem Bildschirm. Geben Sie diese Bildschirmausgabe exakt an.

Kapitel 4

Softwarekonstruktion: Vom Modell zum Produkt



Das Konstruieren von Software ist ein verwobener Prozeß von Analyse- und Synthese-Vorgängen. Ohne eine klare Vorstellung über die gewünschte Situation nach der Übertragung von Arbeit auf Computer(netze) kann die nützliche Software nicht professionell aus Bausteinen, zum Beispiel aus Standardsoftware oder fremdgefertigten Rahmenwerken, zusammengesetzt werden. Die Anforderungen an das Softwaresystem müssen daher hinreichend präzise notiert werden.

Die Wahl der richtigen Bausteinen ist nicht trivial. Einerseits scheint es zunächst eine Vielzahl von möglichen Alternativen zu geben. Andererseits zeigt sich, daß manch dv-technisches Detail die Verwendung des vorgesehenen Bausteines ausschließt oder zumindest wesentlich erschwert. Die Softwarekonstruktion erfordert daher Detailkenntnis über die potentiell geeigneten Bausteine und Werkzeuge.

Als Beispiel für die Lösungsvielfalt betrachten wir die dynamische Erzeugung von Dokumenten in einer Web-üblichen Client-/Server-Architektur. Skizziert werden die Optionen *Common Gateway Interface* (CGI), *Server Side Includes* (SSI), *HTML embedded Scripting Language* und

Java^{TM}-Programme auf der Client-Seite (≡ *Applet*) und auf der Server-Seite (≡ *Servlets*). So wird das Problem der Verteilung von Arbeiten im Computernetz erfahrbar.

Wegweiser

Das Kapitel „Softwarekonstruktion: Vom Modell zum Produkt“ erläutert:

- die Modellierung mit Hilfe einer Gliederung in Syntax, Semantik und Pragmatik,
  ↪ Seite 186 . . .
- die Phasen der Analysen und der Entwürfe mit der Unterscheidung in eine vorgefundene Situation (≡ IST) und eine angestrebte Situation (≡ SOLL),
  ↪ Seite 199 . . .
- die Phase der Implementation mit der Betonung der Relevanz von Details und
  ↪ Seite 201 . . .
- skizziert abschließend im Sinne eines WI-Ausblicks die persönlichen Hoffnungen, Visionen und Pläne.
  ↪ Seite 231 . . .

4.1 Modellierung: Syntax, Semantik, Pragmatik

Jetzt sind wir nicht mehr
abhängig von Königen und Fürsten,
dafür von Softwarekonstrukteuren!

In der Regel wird versucht Software zu konstruieren mit den Merkmalen:

- *Industrial Strength* (≡ für den harten Alltagseinsatz)
- *Resolutely Simple* (≡ keine unnötige Komplextität)
- *Multi-Platform* (≡ für mehrere Betriebssysteme)

Trotz aller Werkzeuge zur Softwarekonstruktion; das heißt trotz CASE-Tools (*Computer Aided Software Engineering*), bleibt das Übertragen von Arbeit auf Computer schwierig. Es wirft viele Probleme auf, zum Beispiel:

- Wie ist ein Modell darzustellen? ⇒ **Syntax** **Syntax**
  Wie kann man ein (Übertragungs-)Modell darstellen, wenn es gleichzeitig Verständigungsmodell für die Menschen und letztlich ausführungsfähige Anweisungen für den Computer sein soll?

- Was wird beschrieben? ⇒ **Semantik**[1] **Semantik**
  Was legt ein solches Modell fest und was nicht? Es ist ja doch „nur“ ein Modell, eine Abstraktion der auszuführenden Arbeit und nicht die Arbeit selbst.

- Was sind die Folgen? ⇒ **Pragmatik** **Pragmatik**
  Was bedeutet eigentlich die Einführung dieser neuen Arbeitsteilung zwischen Mensch und Computer? Unstrittig löst sie eine gewaltige Umstellung aus — betroffen sind organisatorische Strukturen bis hin zur Fähigkeit zu einem sozialen Verhalten in der Arbeit.

4.1.1 Konglomerat von Enscheidungsfragen

Die Entwicklung von Software wirft im besonderen Maße konflikt- und risikobehaftete Entscheidungsfragen auf. Die Mikrosicht betont in der Regel die Erreichung einer hohen Produktivität und Qualität unter Einhaltung vorgegebener Restriktionen (Kosten, Termine). Bei einer Makrosicht ist es die strategische Bedeutung für die Organisationseinheit (Unternehmung, Verwaltung) auf ihrem Wege zu mehr „Wissen“ in einer Informationsgesellschaft. Ob die Softwareentwicklung tatsächlich im Zusammenhang mit einer Massenproduktion von Wissen zu verstehen ist, erscheint fraglich. WI>Dataerläutert nicht weiter, ob mehr Software Ordnung in ein Chaos der Informations-Überschwemmung

Mikrosicht

Makrosicht

[1] Anmerkung: Die Zuordnung von Syntax zu Semantik ist üblicherweise (siehe zum Beispiel UML) nicht präszise formal spezifiziert, sondern besteht weitgehend nur informell. „Der Preis . . . einer informellen Zuordnung von Syntax zu Sematik ist hoch. Denn ob mit einem bestimmten syntaktischen Gebilde der gewünschte Effekt erzielt wird, hängt allein von Geschick und Intuition ab . . . .“ ↪ [Kre03] S. 83.

und -Verseuchung bringt und letztlich mehr „Wissen“ beschert. In WI->Datainteressiert die Softwareentwicklung aus der Perspektive einer Übertragung von Arbeit auf Computer. In diesem Zusammenhang stellen sich die üblichen Fragen nach dem Warum, Wozu, Weshalb usw. Einige dieser sogenannten „*W-Fragen*“ lassen sich wie folgt formulieren:

Was?

Was soll wozu entwickelt werden?

Zu definieren sind Ziele–Zwecke, Nutznießer–Verlierer, Interessengegensätze. Im Mittelpunkt einer Zieldiskussion stehen:

- die **Rationalisierung**,
- die **Qualitätsverbesserung**, zum Beispiel präzisere Karten im Vermessungswesen, und/oder
- **neuartige Leistungen** (Innovationen), zum Beispiel Schwachstellenerkennung durch Zusammenführung großer Datenmengen (*Data Mining* ↪Abschnitt 2.5.2 Seite 120).

Wer?

Wer soll entwickeln?

Führt der Team-Ansatz zum Erfolg? Man kombiniere Spezialisten (Fachexperten) aus dem Arbeitsbereich, in dem die Software eingesetzt werden soll, mit Spezialisten der Softwareentwicklung (Informatiker), gebe ihnen Mittel (Finanzen & Zeit) und erhält dann nach Ablauf der vorgegebenen Projektzeit die gewünschte Software. Wer soll in diesem Projektteam der maßgebliche Entwickler sein – quasi der Architekt des Gebäudes? Eine Fachexperte oder ein Informatiker?

Wie?

Wie soll entwickelt werden?

Wie lautet das Leitmotto? Welches Denkmodell (Paradigma) ist geeignet? Ist Softwareentwicklung die Erzeugung eines „Produktes“, das maßgeblich vom ersten Impuls – dem Auftrag – geprägt ist und dessen Erstellung ein deduktiver Prozeß ist? Lassen sich aus einem Softwareentwicklungsauftrag:

- die Problemstellung präzise ableiten,

- die Softwarekonstruktion Schritt für Schritt vollziehen,
- das Ergebnis prüfen und freigeben?

Handelt es sich um eine klar bestimmbare Aktivitätenfolge, bei der das Ergebnis der vorhergehenden Aktivität den Lösungsraum der nachfolgenden Aktivität einschränkt (definiert)? Sind die einzelnen Phasen und die damit verbundenen Arbeiten im voraus festlegbar? Oder ist das klassische Denkmodell eines linearen Phasendurchlaufes ungeeignet und durch ein neues zu ersetzen? Vielleicht durch ein Denkmodell der permanenten Wissenszunahme, charakterisiert durch das Begriffspaar **lernendes System**.

Ist Softwareentwicklung (nur) eine Ingenieuraufgabe?

Ingenieur-aufgabe?

Was ist anders als bei einer klassischen, mechanischen Ingenieuraufgabe? Was sind zum Beispiel die Unterschiede im Vergleich zum Bau eines Schiffes? Sind signifikante Differenzen feststellbar? Warum nimmt man nicht einfach die bewährten Rezepte aus dem Schiffbau, dem Städtebau etc.? Das Verständnis für die Idee *Software Enginieering*[2] (≈ Softwaretechnik). vermittelt die Etymologie, also die Lehre von der wahren Bedeutung des Wortes, das heißt vom Ursprung des Wortes. „Ingenium“ [lateinisch] verweist auf natürliche Begabung, Scharfsinn und Erfindungskraft, auf Logik und Mathematik. „Ingenieur“ [französich: Kriegsbaumeister] verweist auf Management und Kooperation mit dem Ziel ein größeres Produkt zu schaffen.

Erfordert beispielsweise die Softwareentwicklung für eine programmierte Waschmaschine eine andere **Arbeitstechnik** als die Softwareentwicklung für ein Finanzwesen? Zu klären ist daher, ob trotz aller Unterschiede im Anwendungsbereich dieselben Grundsätze und Methoden einen universellen, tragfähigen Rahmen bilden. Aus der Sicht des Konstrukteurs drängen sich dabei folgende Fragen auf:

- Geht es um eine Übertragung monotoner, klar strukturierter und wohl definierter Prozesse oder um komplexe Prozesse der Informationsverarbeitung, die intelligenten Umgang mit diffusem Wissen erfordert?

[2] Zu den Wurzeln, Stand und Perspektiven der Disziplin *Software Engineering* siehe z. B. ↪ [Br+02].

- Sind die zu programmierenden Abläufe aus manuellen Prozessen (unmittelbar) bekannt oder sind die Abläufe primär kognitive Prozesse und daher nicht direkt beobachtbar?
- Besteht die Verarbeitung primär aus homogenen, strukturierten Massendaten oder hetorogenen, unstrukturierten Wissenseinheiten?
- Entsteht Komplexheit primär aufgrund des Umfanges der Datenmenge oder durch die Reichhaltigkeit der Wissensstrukturen?
- Geht es um relativ wenige Datentypen mit vielen Ausprägungen (Instanzen) eines Typs oder um viele Strukturtypen mit oft wenigen Instanzen eines Typs?

Im Hinblick auf die erforderlichen Arbeitstechniken sind die Größe der Software und ihr Erstallungsaufwand bedeutsam. Die Tabelle 4.1 Seite 191 skizziert benötigte Arbeitstechniken in Abhängigkeit zur Konstruktionsgröße. Dort ist der Umfang des geschätzten Quellcodetextes nur ein grober Maßstab. Die Angabe des Aufwandes in *Manpower*-Jahren (MJ) ist umstritten und berechtigt kritisierbar ('The Mythical Man-Month' [Br75]). Hier dienen die LOC- und MJ-Werte nur zur groben Unterscheidung, ob ein kleines oder großes Team die Aufgabe bewältigen kann. Bei einer größeren Anzahl von Konstrukteueren ist eine größere Regelungsdichte erforderlich. Die Dokumentationsrichtlinien, die Arbeitsteilung und die Vollzugskontrollen sind entsprechend detailliert zu regeln. Es bedarf einer umfassenden Planung und Überwachung des gesamten Lebenszyklus.

Regelungsdichte

4.1.2 „Human Factor"-Dominanz

Der Softwarekonstrukteuer hat (hoffentlich!) von den Technokraten gelernt: *Es klappt nicht!* Durch die feingesponnenen Netze der Netzpläne, Phasenpläne, Entscheidungstabellen und Organigramme entschlüpft immer wieder der Mensch als *nicht programmierbares Wesen*.[3] Die Gestaltung der Mensch-Maschine-Kooperation kann daher nur bedingt Hard-/-Software zentriert erfolgen. Sie muß den Menschen als dominierenden Gestaltungsfaktor, als „*Human Factor*", beachten.

[3] Zur Thematik „Human Factor" hat Helmut Hofstetter als einer der ersten wissenschaftliche Untersuchungen durchgeführt; zum Beispiel ↪[Hof83].

Lfd. (I)	**Konstruktions-Kategorie** (II)	**Konstruktionsgröße** [LOC] (III)	**Aufwand** [MJ] (IV)	**Erforderliche Arbeitstechniken** (Prinzipien, Methoden, Instrumente) zur: (V)
1	Kleine Konstruktion	< 1.000	< 0.5	o funktionalen Strukturierung o Datenrepäsentation
2	Mittlere Konstruktion	< 10.000	< 4	o Projektplanung o Projektüberwachung vom gesamten Lebenszyklus o Anforderungsanalyse (Requirements Engineering) o plus Lfd. 1
3	Große Konstruktion	< 100.000	< 25	o Durchführbarkeitsstudie o Computernetz- und DBMS-Spezifikation o Konfigurationsverwaltung o plus Lfd. 2
4	Sehr große Konstruktion (noch größer „zerfallen“ in eigenständige Teile)	mehrmals 100.000	> 25	o Softwareevolution o Hardwareevolution o Dynamik der Anforderungen o plus Lfd. 3

Legende:

LOC ≡ *Lines of Code* (Länge des Quellcodetextes)
MJ ≡ *Manpower* Jahre

Tabelle 4.1: Konstruktions-Kategorien und erforderliche Arbeitstechniken

Niemand bezweifelt, daß Softwareentwicklungen scheitern (können). Häufig wird die Ursache den Anwendern (Auftraggebern & Benutzern) zugewiesen, weil diese sich nicht präzise artikulieren können; ihre Motive, Wünsche, Interessen, Anforderungen etc. bleiben unklar. Die Softwarekonstrukteuer wünschen sich den „emanzipierten Auftraggeber". Vielfältige Projekterfahrungen zeigen jedoch die Medaille von der anderen Seite:

Softwareliebe

- Der Konstrukteuer ist innig verliebt in sein Produkt, so daß er alle Kritik abwehrt — bis es zu spät ist.
- Der Konstrukteuer fokussiert sich auf rein technische Aspekte und sieht die soziale Seite der Veränderung unzureichend.
- Der Konstrukteuer stellt ein fiktives technisches Optimum über die Akzeptanz und die angemessenen Kosten.
- Dem Konstrukteuer fehlt es an Respekt vor Tradition und Improvisation.

Das eine Softwareentwicklung mit der alleinigen Ausrichtung „Computer gleich universelle Rationalisierungsmaschine" von den Nutzern und Betroffenen selten akzeptiert wird, ist offensichtlich. Darüber hinaus ist insbesondere im sensiblen Bereich des „gläserner Menschen" massiver Widerstand angebracht.

Es ist zu planen, wie eine sinnvolle Zusammenarbeit zwischen Mensch und Computer gestaltet werden kann. Dabei geht es jedoch weniger um die Optimierung des Computers als „Werkzeug". Hilfreicher erscheint eher die aufgeworfenen Fragen aus einem Rollenverständnis einer Kooperation zu betrachten. Die Software sollte daher nicht so konzipiert werden, daß der Mensch in die Rolle eines stupiden Datenzulieferns („Eintippers") und Datenentsorgens („Papierabreißers") gedrängt wird. Die Mensch-Maschine-Kooperation bedarf einer bewußten Gestaltung, insbesondere im Hinblick auf den anzustrebenden Automationsgrad (= Beschäftigungsgrad des Computers). Nicht weil sie softwaretechnisch aufwendig abzubilden ist, sollte eine stupide Aufgabe gleich dem Menschen zugeordnet werden, das heißt in logisch zu Ende gedachter Konsequenz: Es gibt auch eine **abzulehnende Halbautomation oder „Zuwenig"-Automation**.

4.1.3 Grenzen der Softwareentwicklung

Ethik

Das Denkmodell der Mensch-Maschine-Kooperation bedingt, daß Grenzen der Softwareentwicklung und damit des Computereinsatzes zu akzeptieren. Die Grenzen ergeben sich aus aus vielerlei Motiven, zum Beispiel aus der Ethik des Konstrukteurs (Ingenieurs). Zu ziehen sind mindestens folgende Grenzen (↪[Fl85] S. 53):

- *Grenze des fachlich verantwortbaren Computereinsatzes,* das heißt keine Softwareentwicklung für Bereiche bei denen aufgrund eines verfehlten Vertrauens in die Leistungsfähigkeit von Software, unverantwortbare Risiken eingegangen werden.

  fachliche Grenze

- *Grenze des zwischenmenschlich verantwortbaren Computereinsatzes,* das heißt, dort wo Computer aufgrund einer verfehlten Gleichsetzung von Menschen mit Maschinen eingesetzt werden sollen, zwischenmenschlicher Austausch behindert wird, menschliches Erleben verkümmert, menschliche Zuwendung wegfällt und „soziale Netze“ zerstört werden.

  Mensch ≠ Maschine

- *Moralisch/politische Grenzen des Computereinsatzes,* das heißt keine Software, um Computer in die Lage zu versetzen, das zu tun, was ohne Computer nicht gemacht werden darf.

  moralische Grenze

Der Konstrukteuer hat Verantwortung für das zu übernehmen, was er in die Software „gießt“. Er trägt diese Verantwortung und darf sich nicht hinter denjenigen, der den Auftrag erteilt, verstecken.

4.1.4 Problem: Fachsprache⇔Informatiksprache

Die ersten Phasen eines Softwareprojektes, zum Beispiel die Problemanalyse, die Definition der Ziele und Konflikte, die Ist-Aufnahme des System(umfeldes), die Analyse des Bedarfs und der Schwachstellen sind maßgeblich vom Auftraggeber und den späteren Nutzern bestimmt. In den anschließenden Phasen, zum Beispiel des technischen Entwurfs, der Feinplanung, der Programmierung, des Integrationstestes, dominieren die Softwarekonstrukteure. Beide Seiten nutzen – und sind geprägt von – unterschiedlichen Terminologien[4]. Einerseits ist es die Fachsprache

[4] Unter einer Terminologie versteht man allgemein die Gesamtheit der in einem Fachgebiet üblichen Fachwörter und Fachausdrücke und die Lehre von ihnen.

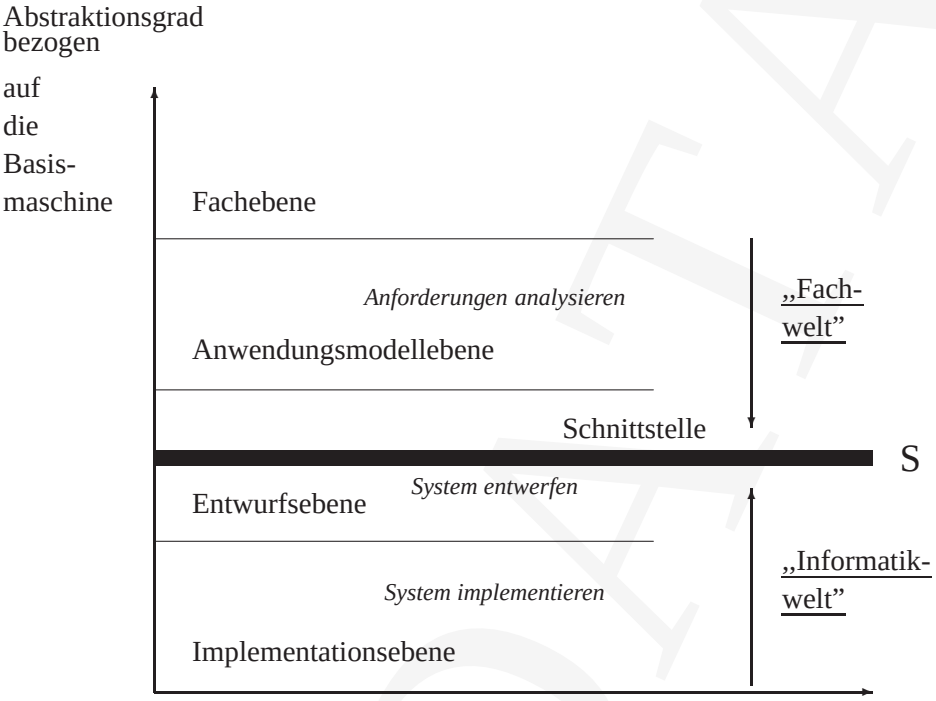


Abbildung 4.1: Terminologie-Problem: Nutzer⇔Softwarekonstrukteur

des Anwendungsfeldes, andererseits ist es die Terminologie der Informatik („Computerwelt"). Bei einer Softwareentwicklung sind Probleme, die ihre Ursache in diesen unterschiedlichen Terminologien haben, zu lösen.

Das Bild 4.1 Seite 194 zeigt vier Abstraktionsebenen: Fachebene, Anwendungsmodellebene, Entwurfsebene und Implementationseben. Sie verdeutlichen, daß bei der Softwareentwicklung ein Übergang von der Fachterminologie („Fachwelt") zur Informatik-Terminologie („Informatik-Welt") erforderlich ist. Mit dem Zwang auf der Ebene der Implementation in der Terminologie der Informatik zu argumentieren und zu formulieren, zeichnet sich rückwärts betrachtet die Notwendigkeit ab, das

Ergebnisdokument der vorhergehenden Phase ebenfalls in der Terminologie der Informatik zu formulieren. Anderernfalls wäre ein Deduktionsprozeß – im Sinne des Phasenkonzeptes – nicht gewährleistet. Ausgehend von der Fachebene einerseits und der Ebene der Implementation anderseits zeichnet sich eine Grenzlinie ab, bei der die beiden unterschiedlichen Terminologien zusammentreffen. Diese Schnittstelle (S in Abbildung 4.1 Seite 194) kann auf zwei grundlegende Weisen gemeistert werden:

gemeinsame Sprache

1. Vorab wird für die Akteure aus dem Fachbereich und der Informatik eine gemeinsame Sprache definiert. Hierzu gehören die Ratschläge, die mit einem gemeinsamen Begriffslexikon operieren und dem Fachbereich zum Beispiel zumuten, Entscheidungstabelle oder Pseudocode zu kennen, also Sprachmittel der Informatik zu verstehen. Dieser Ansatz unterstellt: Wenn die Fachebene ein hinreichendes Verständnis der Informatik gewinnt und die Informatiker ein hinreichendes Wissen über die Softwareaufgabe, dann entsteht eine gemeinsame, allseits verstandene Projektsprache, die die Schnittstelle S leichter überwinden läßt.

***WAS* vom *WIE* trennen!**

2. Die Schnittstelle S wird bewußt herausgearbeitet. Oberhalb (gemäß Abbildung 4.1 Seite 194) wird ausschließlich mit Begriffen des Fachbereichs operiert, unterhalb mit definierten Informatikbegriffen. Die Übersetzung wird als bewußter Akt gestaltet und im allgemeinen von einem Übersetzer, dessen „Muttersprache“ die Terminologie der Informatik ist, vollzogen. Hierzu gehören die Empfehlungen, die ein besonderes Dokument betonen, zum Beispiel ein Pflichtenheft, das von der Terminologie der Informatik frei sein sollte. Getragen werden diese Ratschläge von der Sorge, daß mit der Terminologie der Informatik einhergehend das *WIE*, also die Ausprägung der späteren Lösung einfließt und damit das *WAS* des Fachbereichs frühzeitig beeinflußt (eingeschränkt) wird.

4.1.5 Lasten- & Pflichtenheft

Aussagen über die zu erfüllenden Leistungen eines Softwareproduktes sowie über dessen qualitative und/oder quantitative Eigenschaften werden als Anforderungen (*Requirements*) bezeichnet. Als Beispiel skizziert Tabelle A.14 Seite 270 Anforderungen für ein Diagnoseprogramm

Kriterium	**Lastenheft**	**Pflichtenheft**
Erstellungszeitpunkt:	Definitionsphase	Planungsphase
Intention:	*Stop-or-go*-Frage	Vertragsgrundlage
Detailierungsgrad:	Grobe Übersicht	Umfassende Beschreibung
Hauptakteure:	Auftraggeber Projektleiter (Fachexperte)	Auftraggeber Projektleiter Fachexperte Systemanalytiker

Tabelle 4.2: Anforderungsdokumentation: Lasten- & Pflichtenheft

im Breich Kraftfahrzeugwesen. Je nachdem welcher Lösungsansatz für die Terminologieproblem gewählt wurde, sind diese als freier Text oder stärker formalisiert notiert. Ihre meist verbindliche Fixierung in Form einer Softwareproduktdefinition wird als Lastenheft und/oder Pflichtenheft bezeichnet, insbesondere wenn es als Basis für die Vergabe von Arbeiten an Dritte (zum Beispiel an ein Softwarehaus) dient. Die Begriffe Lastenheft und Pflichtenheft werden im Alltag häufig als Synonyme verwendet. Üblicherweise werden sie jedoch aufgrund ihres Entstehungszeitpunktes und ihres Detailierungsgrades unterschieden (↪ Tabelle4.2 S. 196).

Anforderungen an Anforderungen

Eine brauchbare Beschreibung der Anforderungen hat mindestens folgende Eigenschaften. Sie ist:

1. eindeutig,
2. vollständig,
3. verifizierbar,
4. konsistent,
5. modifizierbar,
6. zurückführbar (*traceable*) und
7. handhabbar (anwendbar in den Realisierungs- und Pflegephasen).

Der Auftragnehmer (das Softwarehaus) übernimmt die Verpflichtung ein Softwareprodukt, mit den spezifizierten Leistungen und Eigenschaften zu liefern. Für den Auftragnehmer ist das Pflichtenheft das Schlüsseldokument. Es bildet die verbindliche Grundlage, um in der Art und Weise eines Deduktionsprozesses die nachfolgenden Phasen zu durchlaufen und gesichert zum fertigen Produkt zu kommen.

Wegen seiner großen Bedeutung enthalten praxisorientierte Empfehlungen häufig einen detaillierten Gliederungsvorschlag für das Pflichtenheft. Die Tabelle 4.3 Seite 198 zeigt einen komprimierten Gliederungsvorschlag. Dieser Vorschlag ist hier nicht als direkt umsetzbares Kochrezept zu verstehen. Er soll vorab Ideen zum Pflichtenheft verdeutlichen. Dabei geht es im wesentlichen um:

- die Aufteilung der Ziele/Zwecke des Produktes in einen Abschnitt *Produktleistungen & Produkteigenschaften* und in einen Abschnitt *Anforderungen an den Realisierungsprozeß*,

- die Einführung von Abgrenzungsaspekten, das heißt um das Beschreiben von Zielen/Zwecken, die mit dem Produkt bewußt nicht erreicht werden sollen,

- das ausführliche Beschreiben der Einsatzbedingungen und des Produktumfeldes, als Darstellung derjenigen Fakten, Sachverhalte etc., die im Rahmen dieser Softwareentwicklung nicht gestaltbar sind, sondern als gegebene Randbedingungen („Sachzwänge“) aufzufassen sind,

- die funktionale Beschreibung als Definition der „Benutzermaschine“,

- das Erwähnen eines Anhanges, in dessen allgemeinen Teil relevante Dokumente (zum Beispiel Herstellerbeschreibungen der Hardwarekomponenten) aufgenommen werden sollen, und

- zum Schluß (leicht abtrennbar!) ein vertraulicher Anhang, der Informationen enthält, die nur einem begrenzten Personenkreis zugänglich gemacht werden sollen (zum Beispiel Betriebsvergleiche, Preisabschläge, Sonderkonditionen etc.).

1. **Ziele & Zwecke des Produktes**
   (a) Ergebnisorientierte Leistungen und/oder Eigenschaften
      i. Mußaspekte
      ii. Wunschaspekte
      iii. Abgrenzungsaspekte
   (b) Realisierungsprozeßbezogene Anforderungen
      i. Mußaspekte
      ii. Wunschaspekte
      iii. Abgrenzungsaspekte
2. **Randbedingungen für das Produkt**
   (a) Produkteinsatz
      i. Anwendungsbereich
      ii. Benutzergruppen
      iii. Betriebsbedingungen
   (b) Produktumfeld
      i. Nutzbare Ressourcen (Soft-/Hardware)
      ii. Produktschnittstellen
      iii. Voraussichtliche Veränderungen
3. **Funktionale Beschreibung** („Benutzermaschine“)
   (a) Bedienungsmaßnahmen der verschiedenen Benutzer
   (b) Normal-, Ausnahme- und Fehlerreaktionen
   (c) Informationsflüsse (Eingaben/Ausgaben)
4. **Anhang**
   (a) Allgemeiner Anhang
   (b) Vertraulicher Anhang

Tabelle 4.3: Software-Produktdefinition

4.2 Analysen und Entwürfe: IST & SOLL

Ohne eine hinreichend präzise Vorstellung von dem „*Was gesollt werden Soll!*, also von der gewünschten Situation nach der Übertragung von Arbeit auf Computer(netze), hier kurz SOLL genannt, läßt sich ein System nicht konstruieren. Dieses SOLL kann ohne eine hinreichend präzise Kenntnis der gegebenen Situation, hier kurz IST genannt, nicht beurteilt werden. Zu erstellen sind daher IST-Analysen und SOLL-Entwürfe. Im Rahmen dieser Arbeiten spielt die Festlegung der funktionalen Anforderungen an das System (*Requirements*) eine große Rolle. Im folgenden werden daher anhand eines Beispiels solche Anforderungen gezeigt. Dabei muß die Beschreibung der Anforderungen selbst Anforderungen wie zum Beispiel „Eindeutigkeit“ erfüllen.

Requirements

Beispiel: Fitness-Studiokette Die Fitness-Studiokette *Workout and Fitness Generators Comp.* plant die neue Studiogeneration *Controlled Fitness* ($\mathcal{CF}$). Die $\mathcal{CF}$-Studios nutzen eine umfassende Computerunterstützung. Der Computereinsatz ermöglicht eine wesentliche Verbesserung des Trainingsservice. Dieser Service umfaßt beispielsweise:

1. Für jeden Kunden kann ein individuell optimiertes Trainingsprogramm erstellt und laufend angepaßt werden.
2. Ein Kunde kann sich jederzeit über seinen Trainingsfortschritt informieren.
3. Ein Kunde kann durch vielfältige Anreize (Gebührenreduktion, öffentliche Anzeige der aktuellen „Ranking“-Liste, automatisches Foto bei persönlicher Bestleistung und vieles mehr) zur intensiveren Benutzung des $\mathcal{CF}$-Studios motiviert werden.

Der erste Schritt für den Entwurf eines $\mathcal{CF}$-Softwaresystems ist die Präzisierung der funktionalen Anforderungen an das System. Dazu werden diese Anforderungen zunächst in Form eines möglichst präzisen und widerspruchsfreien Textes notiert. Um bei den Folgeschritten, zum Beispiel dem Modellieren auf der Basis von Anwendungsfällen (*use cases*), sich auf die einzelne Anforderung beziehen zu können, werden die Anforderungen auch schon in der ersten Textfassung referenzierbar notiert. Dazu wird im folgenden ein einfacher „Präfix“ $L_{m.n}$ mit $m = 1, \ldots, 99$

Referenzierbar

und $n = 1, \ldots, 99$ gewählt.Das L steht für funktionale Leistung. Klar ist, es gibt eine Menge Anforderungen an die $\mathcal{CF}$-Software, die nicht von unserem Typ L sind. Ein Beispiel dafür wäre die leichte Lernbarkeit der Handhabung. Klar ist aber auch, daß die L-Anforderungen die Basis für die anderen Anforderungen bilden.

Präsens

Als Startpunkt für den Entwicklungsprozeß werden daher die funktional-geprägten Anforderungen notiert. Dabei wird unterstellt, daß es das System $\mathcal{CF}$ schon gibt. Die Anforderungen werden dann im Präsens formuliert, das heißt zum Beispiel „$\mathcal{CF}$ verwaltet ...“ und nicht „$\mathcal{CF}$ wird ... verwalten.“. In dieser Form können sie dann später direkt in andere Dokumente (zum Beispiel in das Wartungshandbuch) übernommen werden.

- L_1 $\mathcal{CF}$ verwaltet folgende Daten:
  - $L_{1.1}$ Kundenidentifizierung, Adresse, Geschlecht, Bankabbuchungsgenehmigung
  - $L_{1.2}$ Gebührenforderungen und Zahlungen
  - $L_{1.3}$ IST-Konditionen (Gewicht, Ruhepuls, Blutdruck)
  - $L_{1.4}$ mittel- und langfristige Trainingsziele
  - $L_{1.5}$ erzielte Leistungen pro Sportgerät (SG) und Datum
- L_2 $\mathcal{CF}$ erkennt den Kunden durch seine *Chip Card* (CC)
- L_3 $\mathcal{CF}$ erstellt im Dialog mit dem Trainer für den Kunden dessen Trainingsprogramm (TP) und legt damit die benutzbaren SGs fest.
- L_4 Das TP weist die SOLL-Werte pro SG und Datum aus.
- L_5 $\mathcal{CF}$ aktuallisiert das TP
  - $L_{5.1}$ wöchentlich anhand der erzielten Leistungen und/oder
  - $L_{5.2}$ nach Aufforderung durch den Kunden und/oder
  - $L_{5.3}$ aufgrund der Erkenntnisse des Trainers.
- L_6 $\mathcal{CF}$ kontrolliert anhand der CC und des TP ob der Kunde das jeweilige SG benutzen darf.
- L_7 Das SG zeigt während der Übung auf dem Bildschirm an
  - $L_{7.1}$ die bisher erzielten Leistungen,
  - $L_{7.2}$ die aktuellen Leistungen und
  - $L_{7.3}$ die SOLL-Leistungsdaten des Kunden.

L_8 $\mathcal{CF}$ weist auf einem großen Bildschirm im Fitness-Raum eine nationale Bestenliste aus:

- $L_{8.1}$ pro SG
- $L_{8.2}$ pro Jahr und Geschlecht.

L_9 $\mathcal{CF}$ berechnet die Gebühren benutzungsabhängig,

- $L_{9.1}$ schreibt Rechnungen und
- $L_{9.2}$ überwacht die Forderungen (Bankeinzugsverfahren).

$L_{10.1}$ $\mathcal{CF}$ druckt auf Anforderung des Kunden dessen sämtlichen Daten graphisch aufbereitet aus.

L_{11} Jedes SG hat einen eigenen Computer (SG-CPU) mit Bildschirm, Sensoren für die Leistungserfassung und CC-Leser ($\rightarrow L_2$).

L_{12} $\mathcal{CF}$ fußt auf einer *Client/Server*-Architektur. Verknüpft sind alle SG-CPUs mit einem zentralen Studio-Computer ($\mathcal{CF}$-CPU).

L_{13} Für Bilanzierungs- und Wartungszwecke ist die $\mathcal{CF}$-CPU über das Internet mit dem zentralen *Host* von *Workout and Fitness Generators Comp.* verbunden.

4.3 Implementation: Details sind bedeutsam!

. . . und aus dem Chaos
sprach eine Stimme zu mir:
„Lächle und sei froh,
es könnte schlimmer kommen!“
. . . und ich lächelte und war froh
und es kam schlimmer . . . !
(verbreitete Ingenieurerfahrung)

Mit wenigen Schlagworten läßt sich die gewünschte Implementation eines Systems nicht hinreichend spezifizieren. Gestaltungsrelevante Fakten liegen häufig in Details verborgen. Ihre Berücksichtigung erfordert fundierte Fachkenntisse. Ein objekt-orientiertes Modell mag die Anforderungen einer Fachabteilung gelungen darstellen, scheitert aber beim Implementieren anhand von „kleinen“ Restriktionen der Technik. Das fachspezifische Modell berücksichtigt in der Regel nicht die gestaltungsrelevanten Aspekte

- der *Persistenz*, also der Dauerhaftigkeit eines Objektes sowie dessen Unabhängigkeit vom Erzeugungsprozeß und

Persistenz

- der *Performance*, also der akzeptablen Reaktionsgeschwindigkeit des Systems bei großen Mengen von Objekten.

Performance

Diese beiden Aspekte haben jedoch wesentlichen Einfuß auf die Implementation und bedingen häufig eine aufwendige Anpassung des fachlichen Modells.

Als Beispiel wird eine Web-übliche Client-/Server-Architektur betrachtet. Der Client sei ein Web-Browser auf einem PC mit Internetanschluß, zum Beispiel *Mirosoft's Internet Explorer 5*. Die Serveraufgaben übernimmt ein Web-Server, der das Protokoll HTTP/1.1 versteht, zum Beispiel der *Apache Server*[5] `httpd` (↪Tabelle 4.4 auf Seite 203). Wird im Browser ein bestimmter *Uniform Resource Locator* (URL) angegeben, dann sollen Daten aus einer Datenbank im Browser angezeigt werden. Die betroffenen Daten werden beispielsweise vom DBMS *Oracle* verwaltet.

IE5

Apache

URL

4.3.1 *Common Gateway Interface*

Als Lösungsbasis bietet sich das *Common Gateway Interface* (CGI) an (↪Bild 4.2 auf Seite 204). Das DBMS wird mit dem Web-Server über ein relativ einfaches Skript, das diese standardisiert Schnittstelle nutzt, verknüpft. Bei jeder Anfrage mit der bestimmten URL-Angabe ruft der Web-Server das zugeordnete CGI-Skript auf und veranlaßt dessen Ausführung. Die Achillesferse der CGI-Skriptlösung liegt in der Belastung des Web-Servers und in der Ausrichtung seines Betriebssystems. Jedesmal muß der Web-Server die CGI-Environment setzen, das Skript in den Arbeitsspeicher laden und den CGI-Prozess starten. Die CGI-Lösung funktioniert gut, wenn das Betriebssystem ausgelegt ist für das schnelle Starten von Prozessen und für das optimale Verwalten einer großen Menge von Prozessen.[6] Wenn allerdings die Anzahl der Anfragen steigt, dann wird das Kreieren des Skript-Prozesses zum Flaschenhals, insbesondere bei einem Betriebssystem, das für „*lightweight*

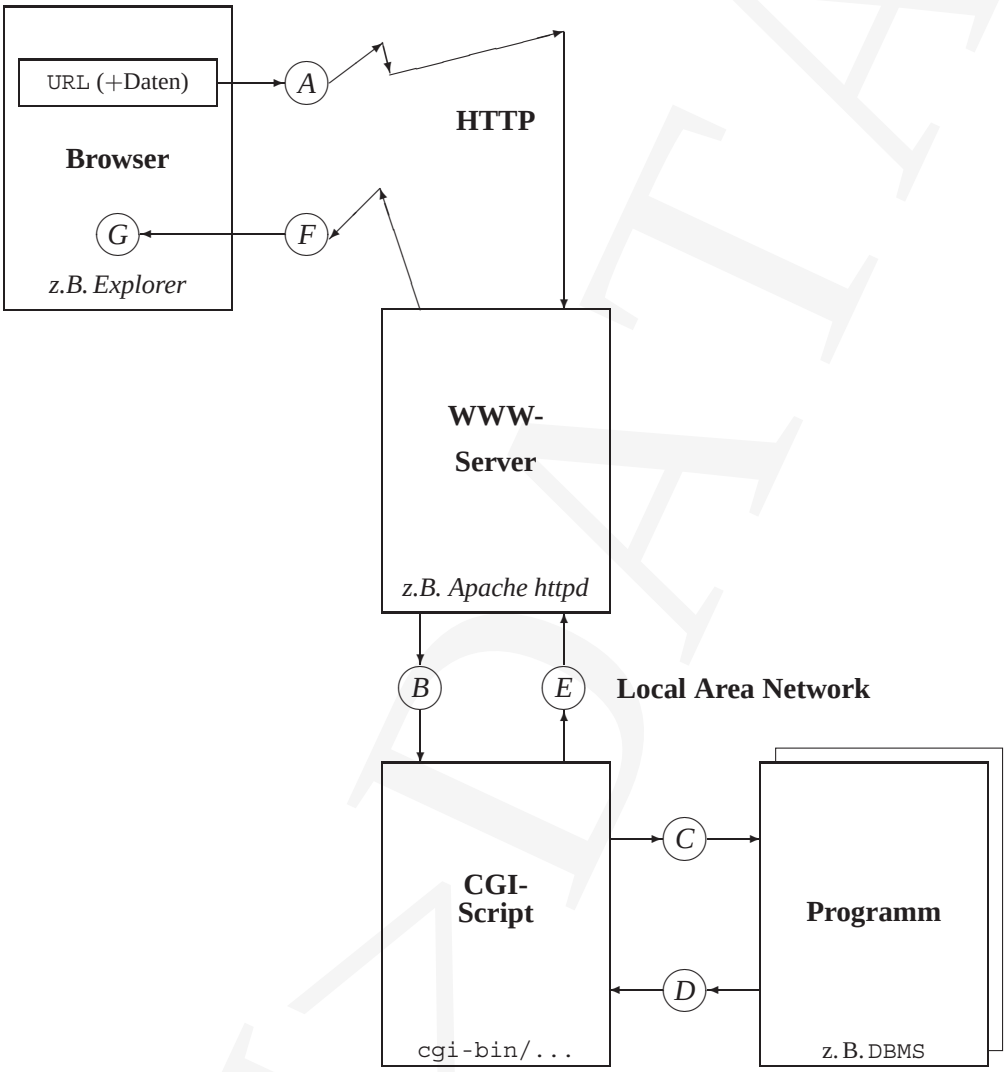
CGI

[5] `http://www.apache.de/` online 07-Sep-1999

[6] Ein Beispiel ist IBM's UNIX-System AIX (*Advanced Interactive Executive*).

```
 1  >telnet as.fh-lueneburg.de 80
 2  Trying...
 3  Connected to as.fh-lueneburg.de.
 4  Escape character is '^]'.
 5  GET /index.html HTTP/1.1
 6  Host: as.fh-lueneburg.de
 7
 8  HTTP/1.1 200 OK
 9  Date: Wed, 22 Sep 1999 06:32:49 GMT
10  Server: Apache/1.3.9 (Unix) PHP/3.0.12 ApacheJServ/1.0
11  Last-Modified: Thu, 16 Sep 1999 09:19:53 GMT
12  ETag: "10a3-85d-37e0b639"
13  Accept-Ranges: bytes
14  Content-Length: 2141
15  Content-Type: text/html
16
17  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
18    "http://www.w3c.org/TR/REC-html40/strict.dtd">
19  <!-- Bonin 08-Sep-1999                        -->
20  <HTML>
21  ...
22  </HTML>
23  GET /content.html HTTP/1.1
24  Host: as.fh-lueneburg.de
25
26  HTTP/1.1 200 OK
27  Date: Wed, 22 Sep 1999 06:33:16 GMT
28  Server: Apache/1.3.9 (Unix) PHP/3.0.12 ApacheJServ/1.0
29  Last-Modified: Mon, 20 Sep 1999 08:00:33 GMT
30  ETag: "106f-8e0-37e5e9a1"
31  Accept-Ranges: bytes
32  Content-Length: 2272
33  Content-Type: text/html
34
35  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
36    "http://www.w3c.org/TR/REC-html40/strict.dtd">
37  <!-- Bonin 08-Sep-1999                        -->
38  <HTML>
39  ...
40
```

Tabelle 4.4: Protokollauszug einer HTTP/1.1-Verbindung zum Web-Server (*Apache*) `http://as.fhnon.de`



Legende:

↪[Bo96]

- *A* *Browser* übergibt dem *Server* einen URL, der auf ein *CGI-Script* verweist.
- *B* *Server* appliziert das CGI-Script und übergibt Daten (Parameter).
- *C* *CGI-Script*-Prozeß ruft das Anwendungsprogramm auf.
- *D* Programm-Output an das *CGI-Script*.
- *E* *CGI-Script* übergibt das aufbereitete Ergebnis als HTML-Dokument an den *Server*.
- *F* *Server* reicht das HTML-Ergebnisdokument an den *Browser* weiter.
- *G* *Browser* formatiert und zeigt die Daten an.

Abbildung 4.2: Kommunikation mit HTTP zwischen Browser und Web-Server mit DBMS

Threads“ konzipiert wurde und bei dem Prozesse zwangsläufig „*heavy-weigth*“ sind.[7]

Zusätzlich wird die CGI-Lösung zum Problem, wenn derselbe Client für die Datenbank mehr als eine Anfrage stellt. Sein CGI-Skript-Prozeß ist nur solange aktiv, wie die aktuelle Anfrage ausgeführt wird. Nach der Anfrage besteht daher die Verbindung zwischen Web-Server und DBMS nicht mehr. Das Anmelden beim DBMS muß daher bei jeder weiteren Datennachfrage wieder neu, also vom erneut geladenen CGI-Skript vorgenommen werden. Mit steigender Anzahl von Anfragen führt dieses Aufgeben der DBMS-Verbindung zum nicht akzeptierbaren Mangel an Performance.

4.3.2 *Application Programming Interface*

Eine Alternative zum CGI-Skript ist das sogenannte Web-Server-API (*Application Programming Interface*). Der API-Mechanismus erlaubt dem Anwendungsprogrammierer die Funktionalität des Web-Servers zu erweitern indem seine eigenen Module direkt zum ausführbaren Serverprogramm[8] gebunden werden. Diese brauchen nur die Transformation von der URL-Angabe zum SQL-Kommando[9] durchzuführen und schon verhält sich der Web-Server wie ein *Front-End* für die Datenbank. Ein zusätzlicher Vorteil ist die Möglichkeiten direkt auf die „internen“ Leistungen des Web-Servers zugreifen zu können und diese darüberhinaus auch neugestalten zu können. Ein erhebliches Risiko ist mit dem API-Konzept verbunden. Ein Fehler in einem hinzugebundenen Modul kann die Web-Server-Leistungen verfälschen oder ihn sogar abstürzen lassen. Zusätzlich besteht ein Portabilitätsproblem. Ein Modul, geschrieben für den Web-Server X, ist selten übertragbar auf den Web-Server Y.

API

URL↪SQL

4.3.3 *Server-Side Includes*

Eine weitere Lösungsmöglichkeit sind sogenannte *Server-Side Includes* (SSI). Dabei werden spezielle Konstrukte in das auszuliefernde HTML-Dokument eingebaut. Benutzt wird dazu das HTML-Kommentar-Konstrukt (↪Tabelle A.15 auf Seite 271). Die speziellen Marken im HTML-

SSI

[7] Ein Beispiel ist Microsoft’s Windows NT.

[8] Serverprogramm ≡ *Server Executable*

[9] SQL ≡ Structured Query Language; Standardsprache für relationale Datenbanken

-Dokument werden vom Web-Server vor dem Ausliefern abgearbeitet. Beispiele hierfür sind Microsoft's *Active Server Pages* (ASP) oder Allaire's *Cold Fusion*[10].

ASP

Mit dem SSI-Konzept wird ein kaum begrenztes Leistungspotential eröffnet, insbesondere wenn die speziellen Kommentarmarken auch Betriebssystemkommandos zur Ausführung bringen. Eine Datei mit dem Suffix `shtml` wird üblicherweise vom Web-Server vor der Auslieferung im Hinblick auf die speziellen Kommentarmarken durchsucht. Das folgende Beispiel `SSI.shtml` (Ergebnis ↪Bild 4.3 auf Seite 208) enthält ein solches Korn-Shell-Kommando[11] nämlich:

```
<!--#exec cmd="ps -ef | wc -l"-->
```

Zunächst wird mit dem `ps`-Kommando jeder laufenden Prozeß als eine Zeile ausgegeben. Mit der *Pipe*-Konstruktion, abgebildet durch den senkrechten Strich „|“, werden diese Zeilen in das Wortzählungsprogramm `wc` transferiert, wobei dessen Parameter `-l` die Zeilen (*lines*) zählt.

Server-Side-Includes-**Beispiel:** `SSI.shtml`

```
 1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
 2    "http://www.w3c.org/TR/REC-html40/strict.dtd">
 3  <!-- Bonin 16-Sep-1999                          -->
 4  <HTML>
 5  <HEAD>
 6  <TITLE>SSI-Beispiel</TITLE>
 7  <LINK HREF="/myStyle.css" REL="stylesheet"
 8    TYPE="text/css">
 9  </HEAD>
10  <BODY>
11  <!--#config errmsg="Leider ein Fehler aufgetreten"-->
12  <!--#config sizefmt="bytes"-->
13  <HR>
14  <H1>Beispiel f&uuml;r
15    <EM>Server-Side Includes</EM>
16  </H1>
```

[10] `http://www.cybergroup.com/html/cold_fusion.html` online 14-Sep-1999

[11] Näheres zur Korn-Shell ↪Abschnitt 3.2 Seite 159.

```
17  <UL>
18    <LI>Dieses angezeigte Dokument
19      <!--#echo var="DOCUMENT_URI"-->
20      ist <!--#fsize file="SSI.shtml"-->
21      Bytes gro&szlig;.
22    <LI>Es wurde zuletzt am
23      <!--#echo var="LAST_MODIFIED"-->
24      ge&auml;ndert.
25    <LI>Es wurde angefragt am
26      <!--#echo var="DATE_LOCAL"-->.
27    <LI>Zum Zeitpunkt der Anfrage
28      liefen auf dem Web-Server
29      <!--#exec cmd="ps -ef | wc -l"--> Prozesse.
30  </UL>
31  <HR>
32  <P class="rechtsKlein">
33    Copyright Bonin Sep-99 all rights reserved
34    <A HREF="http://as.fh-lueneburg.de/">
35    <IMG SRC="/FHNON/home_btn.gif"
36    ALT="[HomPageSymbol]"></A>
37  </P>
38  </BODY>
39  </HTML>
40
```

4.3.4 *PHP Hypertext Preprocessor*

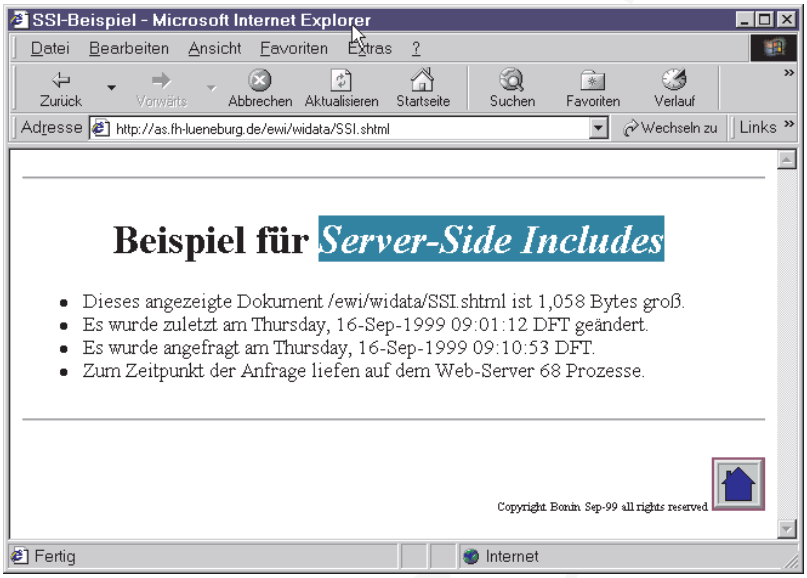
Ein ähnliches Konzept weist der *PHP Hypertext Preprocessor*[12] auf. Das folgende Beispiel `dynamicPage.php3` (Ergebnis ↪Bild 4.4 auf Seite 210) nutzt die Version 3 dieser *HTML embedded Scripting Language*, daher der Suffix `php3` am Dateinamen. Die in HTML einbaubaren PHP-Konstrukte orientieren sich an der C-Syntax und haben die folgende Form:

PHP

`<?php` scripting-code `?>`

Die Konzentration auf die üblichen Anforderungen im Web zeigt zum Beispiel der einfache Zugriff auf HTTP-Daten:

[12] PHP ≡ ursprünglich (1994) von Rasmus Lerdorf *Personal Home Page Tool* genannt. Näheres zu PHP ↪`http://www.php.net/` oder auf dem gespiegelten Web-Server `http://www.php3.de` ; Zugriff 23-Sep-1999.



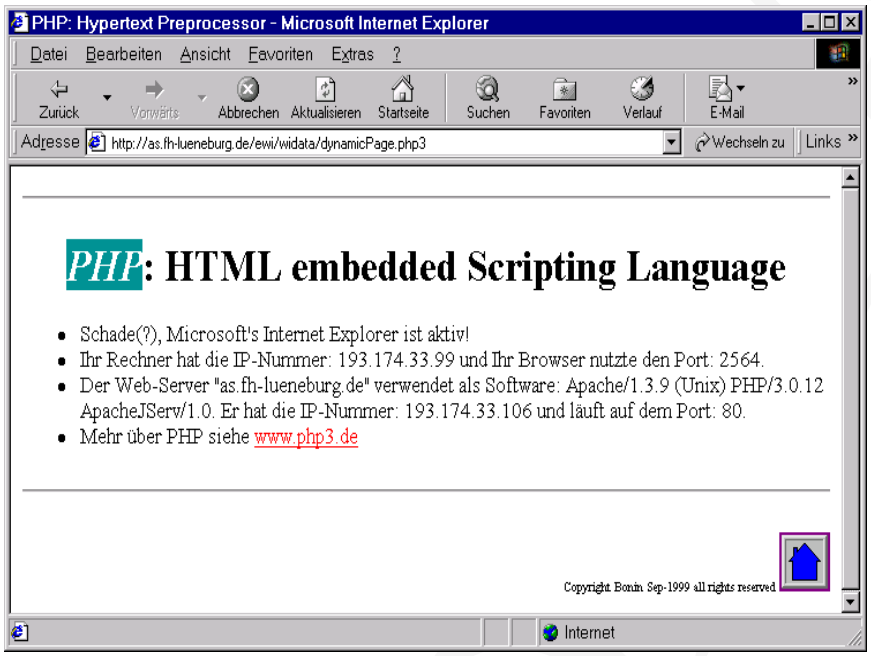
Legende:
↪Quellcodelisting 134 auf Seite 206

Abbildung 4.3: Beispiel für *Server-Side Includes*

```
<?php if(strstr($HTTP_USER_AGENT,"...")) {
          ...
          } else {
          ...
          } ?>
```

PHP-Beispiel: `dynamicPage.php3`

```
 1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
 2    "http://www.w3c.org/TR/REC-html40/strict.dtd">
 3  <!-- Bonin 23-Sep-1999                          -->
 4  <HTML>
 5  <HEAD>
 6  <TITLE>PHP: Hypertext Preprocessor</TITLE>
 7  <LINK HREF="/myStyle.css" REL="stylesheet"
 8    TYPE="text/css">
 9  </HEAD>
10  <BODY>
11  <HR>
12  <H1><EM>PHP</EM>: HTML embedded Scripting Language</H1>
13  <UL>
14    <LI><?php
15          if(strstr($HTTP_USER_AGENT,"MSIE")) {
16            echo "Schade(?), ";
17            echo "Microsoft's Internet Explorer ist aktiv!<BR>";
18            } else {
19            echo "OK, Sie benutzen den Browser ";
20            echo $HTTP_USER_AGENT;
21            echo "!";
22            }
23        ?>
24    <LI>Ihr Rechner hat die IP-Nummer:
25      <?php echo $REMOTE_ADDR;?>
26        und Ihr Browser nutzte den Port:
27      <?php echo $REMOTE_PORT;?>.
28    <LI>Der Web-Server "<?php echo $HTTP_HOST;?>" verwendet
29      als Software: <?php echo $SERVER_SOFTWARE;?>.
30      Er hat die IP-Nummer:
31      <?php echo $SERVER_ADDR;?> und l&auml;uft auf dem
32      Port: <?php echo $SERVER_PORT;?>.
33    <LI>Mehr &uuml;ber PHP siehe
34      <A HREF="http://www.php3.de">www.php3.de</A></P>
```



Legende:
↪Quellcodelisting 135 auf Seite 209

Abbildung 4.4: Beispiel für *PHP HTML Preprocessor*

```
35  </UL>
36  <HR>
37  <P class="rechtsKlein">
38    Copyright Bonin Sep-1999 all rights reserved
39  <A HREF="http://as.fh-lueneburg.de/">
40       <IMG SRC="/FHNON/home_btn.gif"
41       ALT="[HomPageSymbol]"></A>
42  </P>
43  </BODY>
44  <!-- Quelle: /u/bonin/mywww/ewi/widata/dynamicPage.php3 -->
45  </HTML>
46
```

4.3.5 *Servlet*

Im Prinzip wird durch einen *Hypertext Preprocessor* ein HTML-Dokument um die Möglichkeiten einer Programmiersprache erweitert, so daß Variablen, Iterationen und spezielle Konstrukte für den Datenzugriff verfügbar sind. Solche HTML-Einbettungen sind aber nicht standardisiert, so daß sie das Protabilitätsproblem haben. Konsequenterweise wird als Lösung des Problems nicht irgendeine Notation in HTML eingebettet, sondern gleich eine standardisierte Programmiersprache eingesetzt, die vom Web-Server „verstanden“ wird. Ein Weg dazu ist der Einbau eines Sprachinterpreters direkt in den Web-Server. Ein Beispiel für dieses Konzept *Embedded Interpreter* ist der Einbau von **Perl**[13] in den Web-Server Apache mit Hilfe des eingebundenen Moduls *mod_perl*. Ein ähnlicher Ansatz ist ein Web-Server, geschrieben in der gleichen Sprache wie der Interpreter. Ein Beispiel dafür ist Sun Microsystems' *Servlet*-Konzept. Dabei wird die Programmiersprache Java^{TM} genutzt. Sowohl der Web-Server als auch die Zusatzsoftware (anwendungsspezifischen Klassen in Byte-Code) werden von derselben *Java Virtual Maschine* (JVM) abgearbeitet. Ist der Web-Server nicht in Java^{TM} geschrieben, dann wird die JVM in den Web-Server eingebunden, wie zum Beispiel beim Web-Server Apache[14]. Ein einfaches Servlet-Beispiel zeigt der folgende Java^{TM}-Quellcode `Hello.java` (Ergebnis ↪Bild 4.5 auf Seite 214). Die geerbte Java^{TM}-Methode `doGET()` wird durch den HTTP-Aufruf `GET /example/Hello HTTP/1.1` angestoßen. Dabei erzeugt `doGET()` ein HTML-Dokument, welches selbst auf andere Dateien, hier auf die CSS-Datei `myStyle.css` verweist. Dieses Nachladen wird dann vom Browser vollzogen.

Servlet

Servlet-Beispiel: `Hello.java`

```
1  /**
2   * Kleines Beispiel fuer ein
3   * "Servlet"
```

[13] Perl ist eine *Interpreted High-level Programming Language*, die von Larry Wall entwickelt wurde. Nach Aussage von Larry Wall, hat er in Perl alle „*Cool Features*“ eingebaut, die er in anderen Sprachen gefunden hat und alle Features herausgelassen, die nicht *so cool* waren. Näheres zu Perl ↪`http://www.perl.com/` Zugriff 10-Dec-1999.

[14] Web-Server Apache (Version 1.3.9) ist in der Programmiersprache C geschrieben.

```
 4   * Bonin: 22-Sep-1999
 5   *
 6   */
 7  import java.io.*;
 8  import javax.servlet.*;
 9  import javax.servlet.http.*;
10
11  public class Hello extends HttpServlet {
12
13    // Wird ausgefuehrt bei HTTP-GET-Aufruf
14    // (auch HEAD-Aufruf, aber nur Kopfdatenauslieferung)
15    public void doGet (HttpServletRequest request,
16                       HttpServletResponse response)
17      throws ServletException, IOException {
18
19      // Setzen von Header-Felder
20      response.setContentType("text/html");
21
22      // Ausgabe definieren
23      PrintWriter out;
24
25      // Response-Daten schreiben
26      out = response.getWriter();
27      out.println(
28        "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\"");
29      out.println(
30        "\"http://www.w3c.org/TR/REC-html40/strict.dtd\">");
31      out.println(
32        "<!-- Bonin 22-Sep-1999 -->");
33      out.println(
34        "<HTML><HEAD>");
35      out.println(
36        "<TITLE>Servlet-Beispiel</TITLE>");
37      out.println(
38        "<LINK HREF=\"/myStyle.css\" " +
39        "REL=\"stylesheet\" TYPE=\"text/css\">");
40      out.println(
41        "<STYLE type=\"text/css\">");
42      out.println(
43        "BODY" +
44        "{background: url(http://as.fh-lueneburg.de/logo.gif);");
45      out.println(
```

```
46        "text-align: center;}</STYLE>");
47      out.println(
48        "</HEAD><BODY>");
49      out.println(
50        "<H1>Apache JServ Servlet auf " +
51        "<EM>as.fh-lueneburg.de</EM></H1>");
52      out.println(
53        "<P><EM>Hello World</EM></P>");
54      out.println(
55        "<P> Gru&szlig; von as.fh-lueneburg.de</P>");
56      out.println(
57        "<HR><P class=\"rechtsKlein\">Quelle unter: " +
58        "/usr/jserv/ApacheJServ-1.0/example/Hello.java</P>");
59      out.println("</BODY></HTML>");
60      out.close();
61    }
62  }
63  // End of object:
64  // /usr/jserv/ApacheJServ-1.0/example/Hello.java
65
```

4.3.6 JavaScript

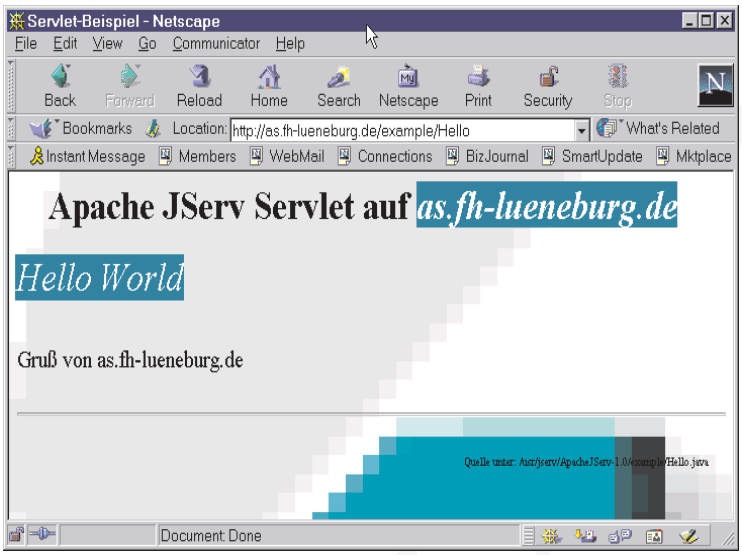
Das Problem der akzeptablen Performance läßt sich prinzipiell auch durch eine Verlagerung von Arbeit auf den Client lösen. Der Browser kann eine Programmiersprache interpretieren und damit die Aufbereitung von Daten selbst übernehmen. *JavaScript*, 1995 eingeführt von der Firma Netscape, und *VBScript*, die Alternative der Firma Microsoft, sind zwei Beispiele für das *Client-Side Scripting*. Die Kombination von diesem Browser-Scripting mit *Cascading Style Sheets*[15] (CSS) und anderen HTML-Erweiterungen bezeichnet man als *Dynamic HTML* (DHTML). Leider wirft DHTML Kompatibilitätsfragen auf, da viele DHTML-Features in den verschiedenen Browser-Typen und -Versionen nicht einheitlich interpretiert werden.

JavaScript

DHTML

Ein einfaches Script-Beispiel für die *Client-Side*-Abarbeitung zeigt das folgende HTML-Dokument `JavaScript.html` (Ergebnis ↪Bild 4.6 auf Seite 217). Es bildet einen simplen Kalkulator ab, der in der Lage ist Grundrechenarten auszuführen. *JavaScript* ermöglicht das Program-

[15] Siehe zum Beispiel Tabelle 4.4 auf Seite 203.



Legende:
↪Quellcodelisting 137 auf Seite 211

Abbildung 4.5: Beispiel für *Java Servlet*

mieren eines Ereignisses, das der Benutzer auslöst. Dazu dient der *Event Handler*. Tabelle A.16 auf Seite 272 skizziert einige *Events*.

JavaScript-Beispiel: `JavaScript.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
  "http://www.w3c.org/TR/REC-html40/strict.dtd">
<!-- Bonin 07-Mar-1997, 24-Sep-1999                    -->
<HTML lang="de">
<HEAD><TITLE>JavaScript-Beispiel</TITLE>
<LINK HREF="/myStyle.css" REL="stylesheet"
  TYPE="text/css">
<SCRIPT type="text/javascript">
<!--
  function berechne(formular) {
    <!-- Zustimmung: confirm() und                      -->
    <!-- Warnung:    alert()                            -->
    <!-- sind eingebaute JavaScript-Konstrukte          -->

    if (confirm("Ist die Eingabe richtig?"))
      <!-- Mit den Feldnamen und den Wort value        -->
      <!-- wird der (eingebene) Wert referenziert.     -->
      formular.ergebnis.value =
        eval(formular.sexpr.value)
    else
      alert("Nun bitte sorgsam korrigieren!.")
  }
-->
</SCRIPT>
</HEAD>
<BODY>
<H1><EM>JavaScript</EM>-Beispiel</H1>
<H2>Kleiner Kalkulator</H2>
<!-- Mit this.form wird -->
<!-- auf das aktuelle Formular verwiesen. -->
<P>
<FORM>
Bitte geben Sie einen einfachen
  mathematischen Ausdruck an:<BR><BR>
<INPUT TYPE="text"   NAME="sexpr" SIZE=20><BR><BR>
<INPUT TYPE="button" VALUE="Ignorieren!"
  ONCLICK="this.form.ergebnis.value = 'Dann eben nicht!'" >
```

```
38  <INPUT TYPE="button" VALUE="Berechnen!"
39    ONCLICK="berechne(this.form)"><BR><BR>
40  Ergebnis der Berechnung: <BR><BR>
41  <INPUT TYPE="text" NAME="ergebnis" SIZE=20
42   ONCHANGE="this.form.ergebnis.value = 'Hier nicht &auml;ndern!'">
43  <BR><BR>
44  </FORM>
45  </P>
46  <HR>
47  <P class="rechtsKlein">Copyright Bonin Mar-1997,...,
48    Sep-1999 all rights reserved
49  <A HREF="http://as.fh-lueneburg.de/">
50       <IMG SRC="/FHNON/home_btn.gif"
51       ALT="[HomPageSymbol]"></A>
52  </P>
53  </BODY>
54  <!-- Quelle /u/bonin/mywww/ewi/widata/JavaScript.html -->
55  </HTML>
56
```

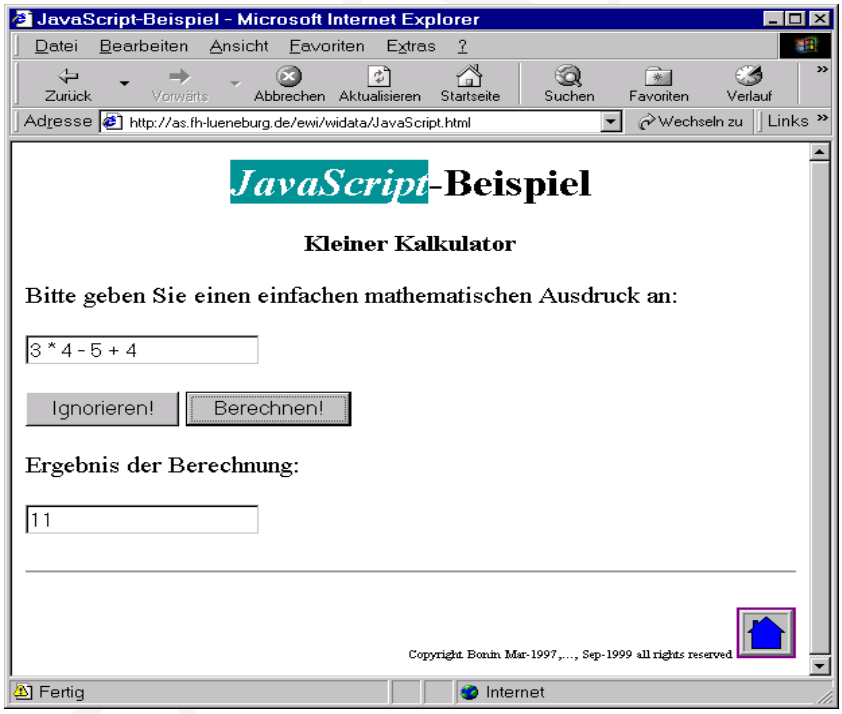
4.3.7 Java^{TM} Applet

Applet

Die Arbeitsteilung zwischen Client (Browser) und Web-Server läßt sich auch durch Austausch von compilierten Programmen organisieren. Der Web-Server schickt dem Browser beispielsweise ein Java^{TM}-Programm, genannt *Applet*, das der Browser auf seiner Maschine startet. Ähnlich funktioniert Microsoft's *ActiveX Technology*. Auch hier werden compilierte Binaries über das Netz transferiert und auf der Client-Maschine vom Browser zur Ausführung gebracht. Ein einfaches Applet-Beispiel für die *Client-Side*-Abarbeitung zeigt das folgende HTML-Dokument `AppletExample.html` (Ergebnis ↪Bild 4.7 auf Seite 221). Dabei wird über das `<OBJECT>`-Konstrukt die Java^{TM}-Klasse `ActionApplet.class` eingebunden. Diese Klasse zeichnet geschachtelte Panels und akzeptiert in der Mitte einen Text (hier: Hello), der in der Statuszeile des Browsers und auf der Console (System.out) ausgegeben wird.

Applet-Beispiel: `AppletExample.html`

```
1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
2    "http://www.w3c.org/TR/REC-html40/strict.dtd">
3  <!-- Testbett fuer Applets                        -->
```



Legende:
↪Quellcodelisting 138 auf Seite 215

Abbildung 4.6: Beispiel für *Java Script*

```
 4  <!-- Bonin  13-Jan-1997; 24-Sep-1999           -->
 5  <HTML lang="de">
 6  <HEAD>
 7  <BASE href="http://as.fh-lueneburg.de/">
 8  <TITLE>Applet-Beispiel</TITLE>
 9  <LINK HREF="/myStyle.css" REL="stylesheet"
10    TYPE="text/css">
11  </HEAD>
12  <BODY>
13  <H1>Applet-Beispiel</H1>
14  <H1>
15  <OBJECT
16    codetype="application/java"
17    codebase="/ActionApplet"
18    classid="java:ActionApplet.class"
19    width="350" height="125"
20    standby="Hier kommt gleich was zum Clicken!">
21    Java Applet ActionApplet.class
22  </OBJECT>
23  </H1>
24  <HR>
25  <P class="rechtsKlein">Copyright Bonin Jan-1997,...,
26    Sep-1999 all rights reserved
27  <A HREF="http://as.fh-lueneburg.de/">
28       <IMG SRC="/FHNON/home_btn.gif"
29       ALT="[HomPageSymbol]"></A>
30  </P>
31  </BODY>
32  <!-- Quelle /u/bonin/mywww/ewi/widata/AppletExample.html -->
33  </HTML>
34
```

Java-Quellcode: `ActionApplet.java`

```
 1  /**
 2   * ActionApplet.class
 3   * Bonin 22-Jan-1997
 4   *  Update 13-Jul-1998
 5  */
 6
 7  import java.awt.*;
 8  import java.applet.*;
 9
```

```
10  public class ActionApplet extends Applet {
11    // Textfeld zur Erfassung eines Textes, der
12    // dann in der Statuszeile des Browsers angezeigt wird.
13    private TextField txt;
14    public void init() {
15      // Initialisierung mit Font Helvetica, Bold, 24
16      Font pFont = new Font("Helvetica",Font.BOLD,24);
17      setLayout(new BorderLayout());
18      setBackground(Color.white);
19      setForeground(Color.green);
20
21      add("North", new Button("Norden"));
22      add("South", new Button("Süden"));
23
24      // Erzeugt ein Panel p0
25      // mit Struktur im Zentrum
26
27      Panel p0 = new Panel();
28      p0.setBackground(Color.red);
29      p0.setForeground(Color.white);
30      p0.setLayout(new BorderLayout());
31      add("Center",p0);
32      p0.add("North", new Button("Oben"));
33      p0.add("South", new Button("Unten"));
34
35      // Erzeugt ein Panel p1
36      // mit Struktur im Zentrum von p0
37
38      Panel p1 = new Panel();
39      p1.setBackground(Color.blue);
40      p1.setForeground(Color.yellow);
41      p1.setLayout(new BorderLayout());
42      add("Center",p1);
43      p1.add("North", new Button("Hamburg"));
44      p1.add("South", new Button("Hannover"));
45
46      // Setzt das Textfeld
47      // in die Mitte des inneren Panels
48
49      txt = new TextField(10);
50      txt.setFont(pFont);
51      p1.add("Center", txt);
```

```
52
53      p1.add("East", new Button("Lüneburg"));
54      p1.add("West", new Button("Salzhausen"));
55
56      p0.add("West", new Button("Links"));
57      p0.add("East", new Button("Rechts"));
58
59      add("West", new Button("Westen"));
60      add("East", new Button("Osten"));
61     }
62    public void start() {
63    }
64    public void stop() {
65    }
66    public void destroy() {
67    }
68    public boolean action(Event evt, Object arg) {
69      System.out.println(
70        ((Button)evt.target).getLabel()
71        + ": " + txt.getText());
72      showStatus(
73        ((Button)evt.target).getLabel()
74        + ": " + txt.getText());
75        return true;
76    }
77  }
78  // Quelle /u/bonin/mywww/ActionApplet/ActionApplet.java
79
```

4.3.8 Helper im Browser

Der Client (Browser) kann so konfiguriert werden, daß beim Aufruf eines Web-Dokumentes, die von dem Web-Server gelieferten Daten in einer vorgegebenen Client-Applikation bearbeitet werden. Wenn beispielsweise der Web-Server ein Dokument im Postscript-Format liefert, dann ruft der Browser das zugeordnete Anzeigeprogramm (*viewer*), zum Beispiel das Programm `GSview`[16] auf. Dabei werden zwei Fälle unterschieden:

[16] Autor: Russell Lang; Quelle für `Ghostscript, Ghostview` und `GSview`: ↪`http://www.cs.wisc.edu/~ghost/index.html` Zugriff 01-Dec-1999

Legende:
↪Quellcodelisting 4.3.7 auf Seite 216

Abbildung 4.7: Beispiel für *Java Applet*

Plug-in: Eine Applikation, die fest in den Browser integriert ist, so daß ihre Ein- und Ausgabe direkt im Browser-Fenster erfolgt.

Helper: Eine Applikation, die vom Browser aufgerufen wird und ihre Ein- und Ausgabe in einem eigenständigen Fenster organisiert.

MIME

Die Zuordnung der Applikation zu einem bestimmten Dokumenttyp erfolgt über den HTTP-Kopf und/oder der Extension des Dokuemtnamens. Der Typ des Dokumentes wird üblicherweise gemäß dem „Standard“[17] MIME (*Multipurpose Internet Mail Extensions*) festgelegt. Zum Beispiel ist bei einem Postscript-Dokument der MIME-Typ: `application/postscript`. Dieser Wert ist entweder im Feld `Content-Type` des HTTP-Kopfes angegeben oder wird aus den Extensionen `ps` oder `eps` geschlossen.[18]

[17] MIME ≡ RFC 1521 (*Request for Comments*), September 1993, Näheres zum Beispiel: ↪`http://www.oac.uci.edu/indiv/ehood/MIME/1521/rfc1521ToC.html` Zugriff 1-Dec-1999

[18] Im Falle des Browser-Zugriffs auf eine lokale Datei gibt es keine HTTP-Kommunikation und damit auch keinen HTTP-Kopf mit dem Feld `Content-Type`.

Diese Browser-Leistung kann man nutzen, um ein bestimmte Applikation auf dem Client mit Daten vom Web-Server auszuführen. Erforderlich sind dazu folgende Festlegungen:

1. eine MIME-Typ
   Zum Beispiel: `application/helperRun`

2. Extensionen der betroffenen Dateinamen
   Zum Beispiel: `myEXE`
   `serverDaten.myEXE` ↪Seite 223

3. die auszuführende Applikation und
   UNIX-Beispiel: Korn-Shell-Script `helperRun.ksh` ↪Seite 222
   Windows-Beispiel: MS-DOS-Shell-Script `helperRun.bat` ↪Seite 223

4. die Übergabe der Web-Server-Daten an diese Applikation.
   Zum Beispiel bei einer Unix-Plattform:
   `/u/bonin/mywww/ewi/helperRun.ksh %s`
   Zum Beispiel bei einer Windows-Plattform:
   `C:\bonin\helperRun.bat %1`
   Hinweis: Die Daten werden hier als Argument in die Applikation eingebracht.

UNIX-Beispiel: Applikation `helperRun.ksh`

```
 1  #!/bin/ksh
 2  #
 3  # @(#) helperRun.ksh dient als Beispiel
 4  # @(#) zur Applikation eines Helpers vom
 5  # @(#) Browser aus.
 6  # @(#) hier: Unix-Plattform
 7  #
 8  # Bonin 02-Dec-1999
 9  #
10  echo "Jetzt kommen die Daten vom Web-Server       "
11  echo "*******************************************"
```

Es muß daher aus der Extension des Dateinamens auf den Dokumenttyp geschlossen werden. Zum Beispiel bei der folgenden Adressenangabe im Browser auf einer UNIX-Plattform: `file://localhost/u/bonin/myDoku.ps`

```
12  cat < $1
13  echo "*******************************************"
14  echo "Daten vom Web-Server beim Client verarbeitet!"
15  #
16  # End of Object /u/bonin/mywww/ewi/helperRun.ksh
17
```

Windows-Beispiel: Applikation `helperRun.bat`

```
 1  @echo off
 2  REM helperRun.bat dient als Beispiel
 3  REM zur Applikation eines Helpers vom
 4  REM Browser aus.
 5  REM hier: Windows-Plattform
 6  REM
 7  REM Bonin 01-Dec-1999
 8  REM
 9  echo "Jetzt kommen die Daten vom Web-Server        "
10  echo "*******************************************"
11  type %1
12  echo "*******************************************"
13  echo "Daten vom Web-Server beim Client verarbeitet!"
14  REM Damit das MS-DOS-Fenster nicht verschwindet
15  pause
16  @echo on
17  REM
18  REM  End of Object C:\bonin\helperRun.bat
19
```

Web-Server-Dokument: `serverDaten.myEXE`

```
 1  <!-- Daten fuer die Verarbeitung mit einem eigenen Helper -->
 2  <!-- Bonin 02-Dec-1999                                     -->
 3  <!-- MIME-Typ: application/helperRun                       -->
 4  <!-- Suffix: myEXE                                         -->
 5  <!-- Web-Server-Dokument:                                  -->
 6  <!-- "http://as.fh-lueneburg.de/ewi/serverDaten.myEXE"     -->
 7  <!-- Unix-Plattform: /u/bonin/mywww/ewi/helperRun.ksh %s   -->
 8  <!-- Windows-Plattform: C:\bonin\helperRun.bat %1          -->
 9  Daten
10  Daten
11  Daten
12  <!-- End of Object: serverDaten.myEXE                      -->
13
```

Option	Performance	Portabilität
CGI	Serverlast, wenig performant	klassich, portabel
Server API	Serverlast: sehr performant	Server spezifisch, kaum portabel
SSI&PHP	Serverlast: performant	Server spezifisch, kaum portabel
Servlet	Serverlast: sehr performant	Server spezifisch, bedingt portabel
Helper/Plug-ins	Clientlast: performant	Browser spezifisch, bedingt portabel
Applet	Clientlast: performant	Browser spezifisch, bedingt portabel

Tabelle 4.5: Lösungsmöglichkeiten im Web

4.3.9 *Client-Server*-Arbeitsteilung im Web

Für die Implementation einer Web-Applikation bieten sich damit folgende Komponenten an:

1. *Server-Side*-Skripten,
2. *Server-Side*-Programme,
3. *Client-Side*-Skripten,
4. *Client-Side*-Programme und
5. ein Mix von allen.

Die Wahl der zweckmäßigen Web-Lösung ist daher nicht trivial, insbesondere wenn *Performance* und *Portabilität* wichtige Anforderungen sind. Tabelle 4.5 auf Seite 224 stellt die skizzierten Web-Möglichkeiten einer Implementation diesen beiden Aspekten stichwortartig gegenüber. Zusätzlich stellt sich bei jeder Arbeitsteilung zwischen Computern die Frage, wer soll die Validierung der Daten vornehmen: Der Server oder der Client? Bei XML-Daten geht man üblicherweise davon aus, daß der Server die Validierung gegenüber der DTD vorgenommen hat. Der Client prüft nur noch die *well-formed*-Korrektheit.[19]

[19] Näheres dazu ↪Abschnitt 2.3.2.

4.4 Übungen

4.4.1 Anforderungen an Anforderungen

Sie notieren in Textform die Anforderungen an ein Softwaresystem, das ein Dritter für Sie konstruieren soll. Welche Anforderungen muß Ihr Text erfüllen?

4.4.2 Warenwirtschaftssystem modellieren

Das Unternehmen *SportwaffenVertriebInternational GmbH* (SVI) setzt pro Geschäftsjahr ≈ 10000 Jagd- und Sportwaffen um. Es werden 11 Zweigstellen beliefert. Die umsatzstärkste Zweigstelle ist in Mannheim. Sie verkauft ≈ 1200 Waffen, die umsatzschwächste ist in Lüneburg und verkauft ≈ 240. Der SVI-Geschäftsführer beauftragt das Softwarehaus *Multimedia InformationsSysteme Tübingen* (MIST-AG) ein modernes Warenwirtschaftssystem grob zu planen. Der Projektleiter Herr Emil Jonnis arbeitet sich in die Materie ein und stellt dabei zunächst folgende Fakten fest:

1. Alle SVI-Produkte sind Sport- oder Jagdwaffen (kurz: Waffen).
2. Es werden Langwaffen von Kurzwaffen unterschieden. Langwaffen sind mindestens 60 cm lang.
3. Eine Waffe ist entweder ein Gewehr oder ein Revolver oder eine Pistole.
4. Gewehre sind Langwaffen. Pistolen und Revolver sind Kurzwaffen.
5. Ein Gewehr ist entweder eine Flinte oder eine Büchse oder eine Kombination davon, also eine kombinierte Waffe.
6. Flinten haben einen glatten Lauf.
7. Büchsen haben einen gezogenen Lauf.
8. Ein Lauf wird durch das Kaliber beschrieben. Die Kaliberangabe ist entstehungsgeschichtlich bedingt. Sie läßt sich als eine Zeichenkette, zum Beispiel für einen Flintenlauf „`12/70`“ oder einen Büchsenlauf „`.308Win`“ beschrieben.

9. Jede Waffe hat eine Herstellernummer. Diese wird vom Hersteller vergeben. Sie ist nur mit dem Herstellernamen eindeutig.
10. Alle Teile, die dem Gasdruck ausgesetzt sind tragen ein Beschußzeichen. Es werden aber nur die Beschußzeichen auf den Läufen im Warenwirtschaftssystem registiert.
11. Es werden derzeit folgende Gewehrtypen verkauft:
    (a) Querflinte ≡ zwei nebeneinanderliegende Flintenläufe
    (b) Bockflinte ≡ zwei übereinanderliegende Flintenläufe
    (c) Bockbüchse ≡ zwei übereinanderliegende Büchsenläufe
    (d) Bockbüchsflinte ≡ ein Flintenlauf liegt über einem Büchsenlauf
    (e) Drilling ≡ eine Querflinte mit zusätzlichem Büchsenlauf
12. Hat das Gewehr mindestens einen Büchsenlauf, dann kann es auch ein Zielfernrohr haben.
13. Ein Zielfernrohr wird durch seine Brennweite und Lichstärke beschrieben.
14. Jedes Zielfernrohr hat zum Zielen ein sogenanntes „Absehen“.
15. Bei den Absehen gibt es unterschiedliche Typen, zum Beispiel Absehen1, Absehen4, Absehen4A oder Absehen8.

UML-Klassendiagramm aufstellen

Herr Emil Jonnis scheint bei dieser Faktenmenge den Überblick zu verlieren. Sie möchtem ihm helfen und entwerfen deshalb ein vorläufiges Klassendiagramm in UML-Notation. Ihr Diagramm sollte möglichst viele der obigen Fakten abbilden. [Hinweis: Da es sich um die fachlichen Klassen handeln sollte, sind „Getter“ und „Setter“ nicht aufzunehmen.]

Modell ergänzen

Herr Emil Jonnis möchte im Rahmen seiner Analyse über Fragen zur Waffenbesitzkarte (WBK) mit Fachleuten diskutieren. Bisher kennt er nur folgende Fakten:

1. Jeder Käufer einer Kurzwaffe muß in seiner gültigen Waffenbesitzkarte den Waffentyp und das Kaliber vorab eingetragen haben.
2. Eine Waffenbsitzkarte wird von der zuständigen Ordnungsbehörde ausgestellt.
3. Jede Waffenbesitzkarte hat bezogen auf die Ausstellungsbehörde eine eindeutige Nummer.
4. Beim Verkauf einer Kurzwaffe wird daher sofort die jeweilige WBK registiert.

Ergänzen Sie Ihr bisheriges Klassendiagramm um diese Fakten.

4.4.3 CGI versus SSI

Vergleichen Sie die beiden Web-Server-Optionen CGI und SSI miteinander. Skizzieren Sie einige Vor- und Nachteile.

4.4.4 Java-Klassen in UML abbilden

Für die Einführung von betriebswirtschaftlicher Standardsoftware gibt es vielfältige Vorschläge zum konkreten Vorgehen. Zum Beispiel empfiehlt die SAP AG, eines der größten Softwarehäuser der Welt, ein Vorgehensmodell für die Einführung ihres Standardpaketes R/3 (R $\equiv$ Real Time). Herr Willi Klugmeier, ein erfahrener Java-Programmierer im Konzeptionsteam der X-GmbH, hat dieses Vorgehensmodell in der Java-Klasse `Vorgehensmodell` abgebildet. Er nutzt dazu seine selbstdefinierte Java-Klasse `Phase`. Im Folgenden sind die beiden Klassen in Form ihrer Quellcodedateien angegeben.

Datei `Vorgehensmodell.java`

```
1  /**
2   *  Modell zum Vorgehen bei Softwareprojekten in Anlehnung
3   *  an die Empfehlung der SAP AG für die Einführung von R/3
4   *
5   *@author     Konzeptionsteam der X-GmbH
6   *@created    23. Juni 2004
7   *@version    1.0 28-Jun-2004
8   */
9  public class Vorgehensmodell
```

```
10  {
11     private Phase organisation;
12     private Phase konzeption;
13     private Phase detaillierung;
14     private Phase realisierung;
15     private Phase produktionsVorbereitung;
16     private Phase produktion;
17
18
19     public Phase getOrganisation()
20     {
21        return organisation;
22     }
23
24
25     public Phase getKonzeption()
26     {
27        return konzeption;
28     }
29
30
31     public Phase getDetaillierung()
32     {
33        return detaillierung;
34     }
35
36
37     public Phase getRealisierung()
38     {
39        return realisierung;
40     }
41
42
43     public Phase getProduktionsVorbereitung()
44     {
45        return produktionsVorbereitung;
46     }
47
48
49     public Phase getProduktion()
50     {
51        return produktion;
52     }
53
54
55     public Vorgehensmodell(
```

```
56           Phase o, Phase k,
57           Phase d, Phase r,
58           Phase pV, Phase p)
59     {
60        organisation = o;
61        konzeption = k;
62        detaillierung = d;
63        realisierung = r;
64        produktionsVorbereitung = pV;
65        produktion = p;
66
67     }
68
69
70     public static void main(String argv[])
71     {
72        Phase p = new Phase(
73              "DokuProzess",
74              "DokuProdukt",
75              "01-Apr-2004",
76              "28-Jun-2004");
77        Vorgehensmodell modell = new Vorgehensmodell(
78              p, p, p, p, p, p);
79        System.out.println(
80              modell.getProduktion().getEnde());
81     }
82  }
83
84
```

Datei `Phase.java`

```
 1  /**
 2   *  Phase für das Vorgehensmodell
 3   *
 4   *@author     X-GmbH Konzeptionsteam
 5   *@created    23. Juni 2004
 6   *@version    1.0 28-Jun-2004
 7   */
 8
 9  public class Phase
10  {
11     String beschreibung;
12     String resultat;
13     String anfang;
14     String ende;
15
```

```
16
17      public String getBeschreibung()
18      {
19         return beschreibung;
20      }
21
22
23      public String getResultat()
24      {
25         return resultat;
26      }
27
28
29      public String getAnfang()
30      {
31         return anfang;
32      }
33
34
35      public String getEnde()
36      {
37         return ende;
38      }
39
40
41      public Phase(
42            String b,
43            String r,
44            String a,
45            String e)
46      {
47         beschreibung = b;
48         resultat = r;
49         anfang = a;
50         ende = e;
51
52      }
53   }
54
55
```

UML-Klassendiagramm

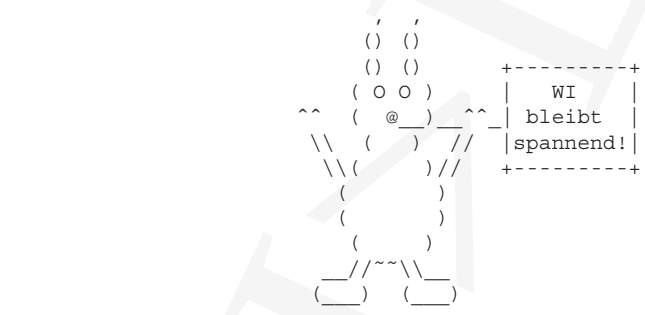
Zeichnen Sie ein entsprechendes UML-Klassendiagramm. Hinweis: Geben Sie in jeder Klasse die Slots (Instanzvariablen), den Konstruktor und

Methoden an. Sogenannte „Getter“ — get-Methoden, wie beispielsweise die Methode `getOrganisation()` — geben Sie dabei nicht an. Zeichnen Sie auch die Verbindung zwischen den beiden Klassen.

Ergebnis der Java-Applikation `Vorgehensmodell`

Die beiden Klassen wurden erfolgreich kompiliert. Geben Sie exakt die Ausgabe auf dem Bildschirm an, wenn die Klasse `Vorgehensmodell` ausgeführt wurde, also folgendes Kommando abgearbeitet wurde:
`java Vorgehensmodell`

4.5 WI-Ausblick: Hoffnungen, Visionen, Pläne



Jeder Text von derartiger Länge und „Tiefe“ verlangt ein abschließendes Wort für seinen getreuen Leser. Es wäre nicht fair, nach so vielen Seiten, die nächste aufzuschlagen und dann den Anhang zu finden. Daher zum Schluß ein kleiner Ausblick. Wie wird sich die WI weiterentwickeln? Was sind unsere Hoffnungen, Visionen und Pläne? Was kennzeichnet die nächste Computergeneration, die der WI neue Impulse gibt?

Wenn die bisher genutzten Generationen schlagwortartig bezeichnet werden als:

1. COBOL-IBM-Mainframe-Generation,
2. UNIX-DEC-Minicomputer-Generation,

3. Windows-Intel-PC-Generation und
4. HTTP-Client/Server-Generation,

wie lautet dann das Schlagwort der nächsten Generation? Vielleicht . . .? Bevor WI>Datazum Science-Fiction-Roman mutiert, zurück zu den WI-Elementen, die in WI>Dataim Mittelpunkt stehen und die Sie nun sicherlich verinnerlicht haben. Für diese lassen sich die folgenden plakativen Thesen formulieren:

Hardware: Alle Technik am Netz
Der Visionär sieht jede Waschmaschine, jede Drehbank, jede Produktionsstation am Netz. Der Produkmanager telefoniert mit seiner „Waschmaschine“ um den aktuellen Zustand von ihr zu erfahren.

Software: Selbstreplizierende Programme
Der Visionär sieht autonome Programme im Netz, die sich durch „Kopulation“ mit anderen Programmen weiterentwickeln. Der Virus von heute wird das Arbeitspferd von morgen.

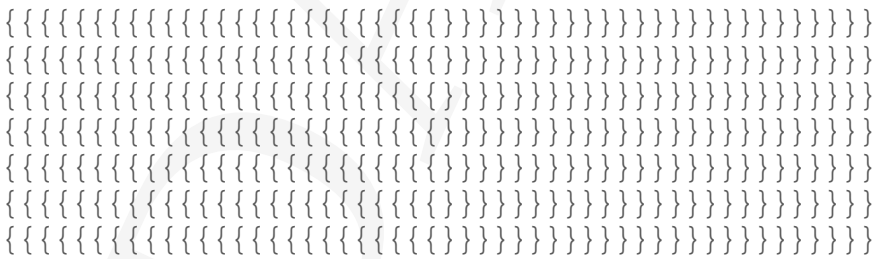
Organisation: Virtuelle juristische Personen
Der Visionär sieht virtuelle Organisationseinheiten, die eine juristische Person sind; bespielsweise ähnlich haften wie eine Aktiengesellschaft und eine steuerliche Ansässigkeit haben.

Nutzer & Betroffene: Mehr Lebensqualität durch Computer
Der Visionär sieht den Computer als Partner zur Schaffung von mehr Lebensqualität. Der Verstärker des Wissens mutiert zum Verstärker der Lebensqualität für Viele.

WI’ler wissen aber, die Dinge müssen sich rechnen. Ob die riesigen Investitionssummen in naher Zukunft auch aufgebracht werden können ist fraglich. Noch sind die „WI-Altsysteme“ nützlich, und damit sicherlich auch die Kenntnisse, die Sie beim Durcharbeiten von WI>Datagewonnen haben. Beim Nutzen Ihrer gewonnenen WI-Kenntisse wünsche ich Ihnen viel Erfolg.

Anhang A

Appendix



A.1 Musterlösungen

Bei den hier angegebenen Lösungen handelt es sich um korrekte Antworten auf die gestellten Fragen. Man beachte aber, daß auch andere Antworten eine korrekte Lösung sein können.

A.1.1 Übung 1.4.1: Hilfsmittelthese

Der Umgang mit Computern verändert die Sicht auf die „reale Welt“ und damit letztlich das Denken des Nutzers — mehr dazu ↪Abschnitt 1.3.4.

A.1.2 Übung 1.4.2: Akronyme DBMS, WAN etc.

DBMS	≡	Datenbankmanagementsystem — eine Software zur Verwaltung großer Datenmengen; Behandlung von konkurrierenden Zugriffen (*Dead-Lock*-Auflösung).
GUI	≡	Graphical User Interface (Tool) — Software zur Gestaltung der Präsentation von Daten auf einem Bildschirm und zur Steuerung der Interaktionen über Tastatur und Maus.
LAN	≡	Local Area Network — stellt Kummunikationskanäle zwischen Computern, die sich alle in einem Gebäudekomplex befinden, bereit.
WAN	≡	Wide Area Network — stellt Kommunikationskanäle zwischen Computern, die sich auf unterschiedlichen Gründstücken befinden, bereit.
XML	≡	Extensible Markup Language — Auszeichnungssprache gemäß SGML; Meta-Grammatik für kontextfreie Grammatiken.
DTD	≡	Document Type Definition — Richtlinien (Grammatik), die ein Dokument dieses Typs erfüllen muß.
`#PCDATA`	≡	Parsed Character Data — Typerklärung von Daten, die nur aus vorgegebenen Textzeichen bestehen.
HTML	≡	Hypertext Markup Language — die Auszeichnungssprache für Web-Dokumente.
UML	≡	Unified Modeling Language — eine Sprache für die Modellierung eines Systems von kommunizierenden Objekten.

A.1.3 Übung 1.4.3: Plattform

```
<!ELEMENT PLATTFORM (C, BETRIEBSSYSTEM, TOOL+)>
```

A.1.4 Übung 1.4.4: Logistik

Mit *Logistik* bezeichnet man die Planungs-, Durchführungs-, Überwachungs- und Kontrollprozesse des wirtschaftlichen Einsatzes und der wirtschaftlichen

Lagerung von Fertigprodukten, Halbfertigprodukten und/oder von Rohmaterial.

A.1.5 Übung 1.4.5: „EVA“-Modell

Das „EVA“-Modell umfaßt die drei Phasen: Dateneingabe, Datenverarbeitung und Datenausgabe. Bei der Datenverarbeitung handelt es sich um die Modifikation von Speicherinhalten.

A.1.6 Übung 2.6.1: Binäraddition

Dezimal-darstellung	Binärwert mit Vorzeichen	Zweierkomplement
-14	10001110	14_{10} als positive Zahl ≡ 00001110
		bitweise Negation davon ≡ 11110001
		Inkrement ≡ 00000001
		11110010
-12	10001100	12_{10} als positive Zahl ≡ 00001100
		bitweise Negation davon ≡ 11110011
		Inkrement ≡ 00000001
		11110100
		11110010 + 11110100 ≡[1]11100110
		Rückrechnung -1 ≡ 11100101
		Negation = Binärwert mit Vorzeichen ≡ 10011010
		Probe: $2^8 - 26_{10} = 256_{10} - 26_{10}$
		$= 230_{10}$
		$= 128_{10} + 64_{10} + 32_{10} + 4_{10} + 2_{10}$
		≡ 11100110
-26	10011010	

A.1.7 Übung 2.6.2: Regel von de Morgen

$\overline{(a \wedge b) \vee c} \parallel (\overline{a} \vee \overline{b}) \wedge \overline{c}$
mit:
$x = (a \wedge b)$
$\overline{x \vee c} \parallel (\overline{a} \vee \overline{b}) \wedge \overline{c}$
mit der Regel von *de Morgan*:
$\overline{x} \wedge \overline{c} \parallel (\overline{a} \vee \overline{b}) \wedge \overline{c}$
mit der Regel von *de Morgan*:
$\overline{x} = \overline{(a \wedge b)} = (\overline{a} \vee \overline{b})$
eingesetzt für $\overline{x}$
$(\overline{a} \vee \overline{b}) \wedge \overline{c} \equiv (\overline{a} \vee \overline{b}) \wedge \overline{c}$ — also korrekt!

A.1.8 Übung 2.6.3: Kommandosequenz

Übung 2.6.3.1

Die Farbe von FOO ist Rot.

Übung 2.6.3.2

$x = -109$ und $y = -118$

A.1.9 Übung 2.6.4: Struktur und Darstellung

Übung 2.6.4.1

Zeile 45 Die `</H2>`-Marke ist hier falsch. Es ist eine `</H1>`-Marke erforderlich.

Zeile 55 Die `</A>`-Marke fehlt.

Zeile 61 Die `</UL>`-Marke ist hier falsch. Es ist eine `</OL>`-Marke erforderlich.

Übung 2.6.4.2

Das *Style-Sheet*-Konzept ermöglicht die strikte Trennung der Spezifikation der Präsentation („Layout“) von der Spezifikation der Struktur des Dokumentes. Die Präsentation läßt sich für viele Dokumente an einer (zentralen) Stelle (Style-Sheet-Datei) gestalten und anpassen. Man spart daher Pflegaufwand und sichert ein einheitliches Layout für die gesamte Publikation. Zusätzlich hat das Konzept der *Cascading Style Sheets* den Vorteil, daß die wirksame Präsentationsspezifikation sich an Prioritäten orientiert. Diese sind hierarchisch festgelegt. Beispielsweise haben lokale Spezifikationen Vorrang vor globalen ($\equiv$ lokales Überschreiben von importierten Spezifikationen). Auch wird so möglich, daß der „Dokumentautor“ die Spezifikation des „Lesers“ überschreibt (oder auch umgekehrt).

Übung 2.6.4.3

Die Darstellung der korrigierten Datei `phasen.html` in einem Standardbrowser zeigt Bild A.1 auf Seite 238.

A.1.10 Übung 2.6.5: Validieren

Übung 2.6.5.1

Die interne DTD wird von `HTMLSyn.xml` erfüllt.

Übung 2.6.5.2

Ein Attribut ohne Wertzuweisung sollte nicht möglich sein. Daher ist vereinfachend für das Element `ATTRIBUT` zu definieren:

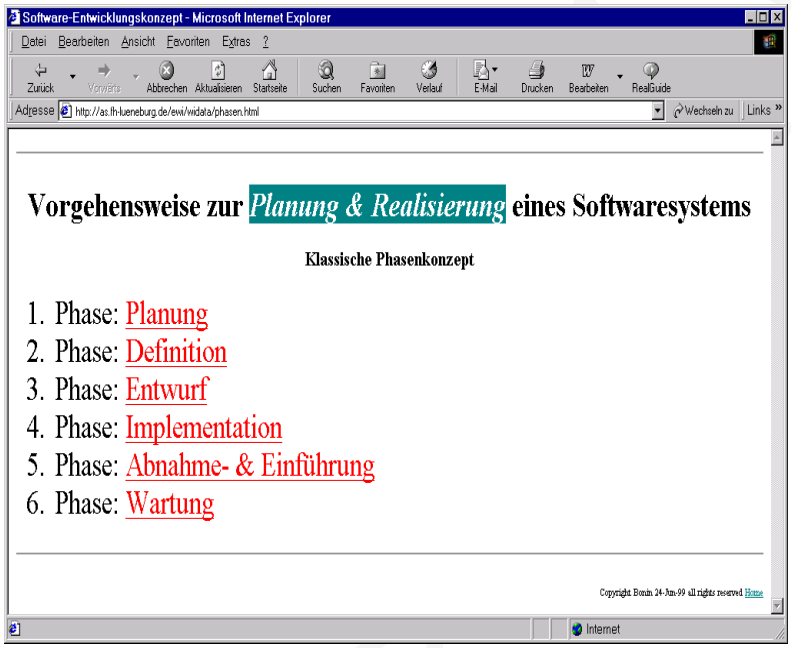


Abbildung A.1: HTML-Dokument `phasen.html` mit *Cascading Style Sheet*

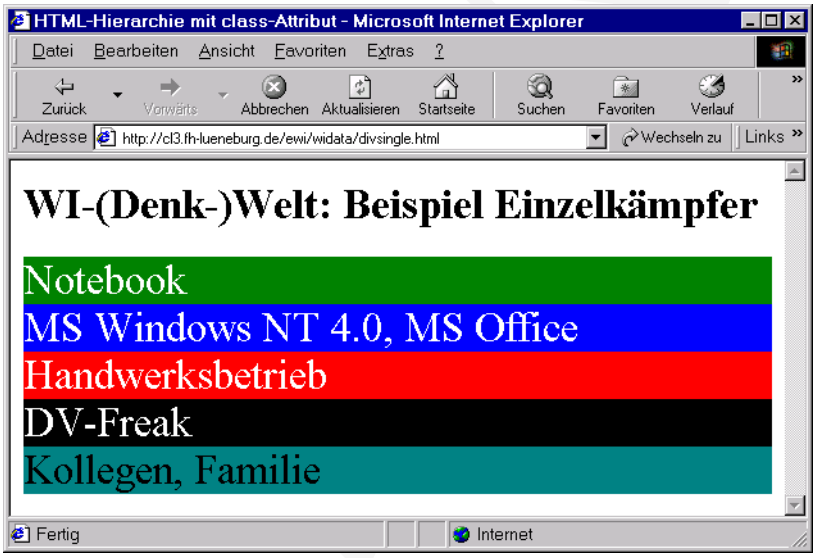


Abbildung A.2: Elementebeispiel für die WI-Welt eines Einzelkämpfers — in HTML mit dem `class`-Attribut

```
<!ELEMENT ATTRIBUT (ATTRIBUTNAME, GLEICH, VALUE)>
```

A.1.11 Übung 2.6.6: Baumstruktur

Übung 2.6.6.1

Die Darstellung der Datei `divsingle.html` in einem Standardbrowser zeigt Bild A.2 auf Seite 239.

Übung 2.6.6.2

Das HTML-Konstrukt:

`<DIV class="bezeichner">...</DIV>`

entspricht dem XML-Konstrukt:

`<bezeichner>...</bezeichner>`.

Beide sind geeignet in ihrem Dokument hierarchische Strukturen abzubilden und dabei semantisch zweckmäßige Bezeichnungen zu benutzen. Die Vorteile von XML kommen erst bei einer komplexen Verarbeitung (mit XSL) zum Tragen; beispielsweise wenn es darum geht, Reihenfolgen zu ändern oder Abkürzungen aufzulösen.

A.1.12 Übung 2.6.7: `Manuscipt.xml`

Übung 2.6.7.1

Ja, die Datei `Manuscript.xml` enthält korrekte Blockstrukturen und wird vom Browser *Microsoft Internet Explorer 5* angezeigt und ausgedruckt.

Übung 2.6.7.2

Zeile 58 Der `Paragraph` mit `label="AOP.aspectJ"` hat keinen korrekten Wert beim Attribut `version`.

Zeile 64 Dem `Chapter` mit `title="References"` fehlt das Attribut `label`.

A.1.13 Übung 2.6.8: `RubyScripting.xml`

Übung 2.6.8.1

Ja, die Datei `Manuscript.xml` enthält korrekte Blockstrukturen und wird vom Browser *Microsoft Internet Explorer 6* angezeigt und ausgedruckt.

Übung 2.6.8.2

Zeile 69 Der `Paragraph` mit `label="RI.copyright"` hat keinen korrekten Wert beim Attribut `version`.

Zeile 76 Dem `Chapter` mit `title="References"` fehlt das Attribut `label`.

A.1.14 Übung 2.6.9: Web-Page für *Jung & Müller* erstellen

Übung 2.6.9.1

```
 1  <?xml version="1.0" encoding="utf-8" ?>
 2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 3    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
 4  <!-- Jung & Mueller 19-Jan-2004  -->
 5  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de">
 6  <head>
 7  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
 8  <meta name="robots" content="index,follow" />
 9  <link href="partner.css" rel="stylesheet" type="text/css" />
10  <title>Gesch&auml;ftspartner der Jung &amp; M&uuml;ller AG</title>
11  </head>
12  <body>
13  <h1 class="large">Gesch&auml;ftspartner der</h1>
14  <h1 class="large"><a href="http://jung-mueller.de">Jung
```

```
15    &amp; M&uuml;ller AG, L&uuml;neburg</a></h1>
16  <ol class="normal">
17    <li>Partner: <a href="http://kirchen-studio.de">KirchenStudio
18      GmbH, Magdeburg</a></li>
19    <li>Partner: <a href="http://glaserei-otto.com">Glaserei
20      Otto AG, Bremen</a></li>
21    <li>Partner: <a href="http://glas-restauration.de">Glas &amp;
22      Restauration AG, Freiburg</a></li>
23    <li>Partner: <a href="http://alte-glaeser.de">Alte
24      Gl&auml;ser GmbH, Karlsruhe</a></li>
25  </ol>
26  <p class="small">Bei Fragen wenden Sie sich bitte an
27  Web-Master <a href="mailto:schmitt@jung-mueller.de">Norbert Schmitt</a></p
28  </body>
29  </html>
30
```

Übung 2.6.9.2

```
 1  /* Cascading Style Sheet: meyer.css  */
 2  /* Jung & Mueller AG    19-Jan-2004  */
 3  .small {
 4    font-size:  10pt;
 5    color:      yellow;
 6    background: black;
 7  }
 8  .normal {
 9    font-size:  14pt;
10  }
11  .large {
12    font-size:  24pt;
13  }
14  body {
15    color:      black;
16    background: yellow;
17  }
18
```

Übung 2.6.9.3

Durch die Trennung der Spezifikation der Präsentation (*Layout*) von der Spezifikation der Struktur & des Inhalts des Dokumentes entstehen folgende:

Vorteile:

1. Gewährleistung eines einheitlichen *Layouts* über eine Menge von Dokumenten
2. Verminderte Wartungsaufwand
3. Nutzung der Vererbung von Styles (lokale `<style>`-Konstrukt überschreibt globales (importiertes) Konstrukt)

Nachteile:
1. Extra Zugriff für den Browser auf die CSS-Datei — Performance!
2. Eine große Menge von `class`-Konstrukten entstehen auch wenn diese zum Teil nur in einem Dokument (einmal) genutzt werden.

A.1.15 Übung 2.6.10: Web-Page für *Wundort* erstellen

Übung 2.6.10.1

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- Gemeinde Wundort 28-Jun-2004  -->
<html xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="de">
<head>
<meta http-equiv="Content-Type"
  content="text/html; charset=utf-8" />
<meta name="robots" content="index,follow" />
<link href="waldberg.css" rel="stylesheet"
   type="text/css" />
<title>Gemeinde Wundort</title>
</head>
<body>
<h1 class="large">Gemeinde Wundort im Landkreis
  <a href="http://waldberg.de">Waldberg</a></h1>
<h1 class="large">Verwaltung</h1>
<ol class="normal">
  <li>B&uuml;rgermeister
  <a href="http://chef.wundort.waldberg.de">
    Herbert Rege</a><br />
    Tel.: 04131/6666 oder 0166666666</li>
  <li>Fachbereich Ordnungswesen: Amtsleiter
  <a href="mailto:ordnung@verwaltung.wundort.waldberg.de">
    Franz Fuchs</a></li>
  <li>Fachbereich Finanzwesen: K&auml;mmerer
  <a href="mailto:finanzen@verwaltung.wundort.waldberg.de">
   Willi Armmaus</a></li>
</ol>
<p class="small">Bei Fragen wenden Sie sich
  bitte an den Web-Beauftragten
  <a href="mailto:web@wundort.waldberg.de">
  Amtsrat Karl Meyer</a>
</p>
</body>
</html>

```

Übung 2.6.10.2

Zeile: 12

```
<style type="text/css">
body {
  color:      blue;
  background: yellow;
}
</style>
```

Übung 2.6.10.3

Vorteile:
- Einheitlichkeit: einheitliche Interaktionsstruktur und einheitliches Layout über alle Seiten / Dokumente
- Handhabung: einfache Benutzung für die Beitragslieferanten („Redakteure"); beispielsweise keine XHTML-Kenntnisse erforderlich

Nachteile:
- Implementations- & Wartungsaufwand
- Ressourcenbedarf

A.1.16 Übung 2.6.11: `Modellierung.xml`

Übung 2.6.11.1

`Modellierung.xml` ist nicht well-formed, da nach Zeile 36 die Endmarke `</Formalismus>` fehlt. Die Datei wird daher nicht angezeigt und kann auch nicht ausgedruckt werden.

Übung 2.6.11.2

Zeile 36: danach fehlt `</Formalismus>`

Zeile 38 und Zeile 44: `label="m1"`
Attribut `label` enthält gleiche Werte, obwohl es in der DTD als Typ `ID` definiert ist.

Zeile 55: Attributwert `"Fertig"` entspricht nicht den vorgegebenen Werten in der DTD.

A.1.17 Übung 3.3.1: Ablaufstruktur

Übung 3.3.1.1

$FOO(1, 2, 3) \rightarrow$
$x = 6$, $y = 10$, $z = 5$ und $r = 7$

Übung 3.3.1.2

Folgende elementare Konstrukte enthält die Funktion $FOO(x, y, z)$:
Sequenz, Alternative, Iteration (`while` und `until`) und Nebenläufigkeit (*concurrency*).

Übung 3.3.1.3

```
 1  <?xml version="1.0" encoding="ISO-8859-1"?>
 2  <!-- Bonin: 18-Nov-1999                    -->
 3  <SEQUENZ>
 4    <STATEMENT />
 5    <ALTERNATIVE>
 6     <PRAEDIKAT>
 7       <STATEMENT />
 8     </PRAEDIKAT>
 9     <TRUE>
10       <SEQUENZ>
11         <UNTIL>
12           <BODY>
13             <SEQUENZ>
14               <STATEMENT />
15               <STATEMENT />
16             </SEQUENZ>
17           </BODY>
18           <PRAEDIKAT>
19               <STATEMENT />
20           </PRAEDIKAT>
21         </UNTIL>
22         <WHILE>
23           <PRAEDIKAT>
24             <STATEMENT />
25           </PRAEDIKAT>
26           <BODY>
27             <SEQUENZ>
28               <STATEMENT />
29               <STATEMENT />
30             </SEQUENZ>
```

```
31            </BODY>
32          </WHILE>
33        </SEQUENZ>
34      </TRUE>
35      <FALSE>
36        <SEQUENZ>
37          <STATEMENT />
38          <STATEMENT />
39          <STATEMENT />
40          <STATEMENT />
41        </SEQUENZ>
42      </FALSE>
43    </ALTERNATIVE>
44    <CONCURRENCY>
45      <STATEMENT />
46      <STATEMENT />
47      <STATEMENT />
48    </CONCURRENCY>
49  </SEQUENZ>
50  <!-- End of object FOO.xml                -->
51
```

A.1.18 Übung 3.3.2: Script-Analyse

Übung 3.3.2.1

```
$ cat < Quartal0199.txt
100.00
200.00
300.00
800.00
700.00
100.00
400.00
600.00
500.00
```

Übung 3.3.2.2

```
$ quartal.ksh
Quartal0199.txt wurde erzeugt!
Anzahl der Positionen: 9
$
```

Übung 3.3.2.3

```
$ what quartal.ksh
quartal.ksh:
   quartal.ksh sammelt Verkaufspositionen
   Syntax: quartal.ksh
$
```

A.1.19 Übung 3.3.3: PGP

Übung 3.3.3.1

Die *LISTIG* GmbH kann ihre elektronische Post signieren und verschlüsseln. Dazu generiert sich jeder Postbenutzer seinen privaten, streng vertraulichen Schlüssel und den zugehörenden öffentlichen Schlüssel; beispielsweise mit Hilfe des Programmes PGP (*Pretty Good Privacy*) von Philip Zimmermann (*Network Associates, Inc.*).

Übung 3.3.3.2

Ein „Key Server“, präziser formuliert, ein *Certificate Server* verwaltet den öffentlichen Schlüssel und die Unterzeichner dieses Schlüssels. Der private Schlüssel darf nicht auf einem solchen Server verwaltet werden. Er ist für andere unzugänglich aufzubewahren.

Übung 3.3.3.3

Bei einer FTP-Verbindung benötigt der Absender eine gültiges Login (USER-ID & PASSWORD) auf dem Computer des Adressaten. Der Adressat muß also eine entsprechende Berechtigung eingeräumt haben; zumindest eine allgemein bekannte Gastkennung. Für eine *Email* benötigt der Absender keine derartige Berechtigung. Er kann an jede gültige *Email*-Adresse seine Daten schicken. Ein Umstellen auf eine `ftp`-basierte Kommunikation kommt wegen der Einräumung entsprechender Berechtigungen daher nicht in Betracht.

A.1.20 Übung 3.3.4: Client⇔Server

Übung 3.3.4.1

Bei einer *Client-Server*-Architektur wird eine Arbeitsteilung zwischen mehreren Computern in einem Netz organisiert. Ein *Client*-Computer gibt einen Auftrag an einen *Server*-Computer, der nach Auftragsausführung dem *Client*-Computer das gewünschte Ergebnis übermittelt. Dabei kann der *Server*-Computer zur Auftragsabarbeitung selbst die Rolle eines *Client* annehmen und einen

Auftrag an einen anderen *Server*-Computer vergeben. Gebräuchlich ist ein 3-Ebenen-Modell: Dateneingabe- & Datenausgabeebene („Präsentationsebene"), Applikationsebene und Datenbankebene. Bezogen auf die *ML-AG* lassen sich die Adressen der Geschäftspartner wie folgt bereitstellen:

Präsentation Ein Standard-PC an jedem Arbeitsplatz mit einem aufgabenspezifischen Erfassungs- und Darstellungsprogramm für die Daten der Nachfolger von *E-SYS* und *M-Access*.

Applikation Je ein eigener *Server* für die Nachfolger von *E-SYS* und *M-Access*.

Datenbank Ein Server für die Daten (Adressen der Geschäftspartner) und das *Datenbankmanagement System* (DBMS); zum Beispiel für die Software ADABAS (Software AG), DB2 (IBM), Oracle oder POET.

Übung 3.3.4.2

Damit die Datenpräsentation und Dateneingabe über einen marktüblichen Standardbrowser erfolgen kann, ist ein Web-*Server* zwischen den *Client*-Computer und den Applikations-*Server* zu schalten. Wenn der Browser die Adresse des Geschäftspartners mit der Indentifikation `musterpartner` anfragt, dann werden folgende Schritte durchlaufen:

1. Der Browser sendet eine Anfrage mit dem Wert `musterpartner` mit den Web-*Server*. Die Kommunikationsbasis ist dabei das *Hypertext Transfer Protocol* (HTTP).
2. Der Web-*Server* erkennt, daß eine Applikation auszuführen ist und gibt die Anfrage an den entsprechenden Applikations-*Server* weiter. Diese Kommunikation kann beispielsweise in der Java^{TM}-Welt realisiert sein und daher auf *Remote Method Invocation* (RMI) basieren.
3. Der Applikations-*Server* erkennt die Notwendigkeit den DB-*Server* anzufragen. Dazu kann beispielsweise ein SQL-Kommando (*Structured Query Language*) an den DB-*Server* gesendet werden.
4. Der DB-*Server* arbeitet das SQL-Kommando ab indem die Adressendaten von `musterpartner` aus dem Gesamtbestand der Geschäftspartner selektiert werden.
5. Diese Adressendaten werden vom DB-*Server* an den Applikations-*Server* und von diesem nach seiner Bearbeitung an den Web-*Server* weitergereicht.
6. Der Web-*Server* schickt das Ergebnis des Applikations-*Servers* an den Browser.

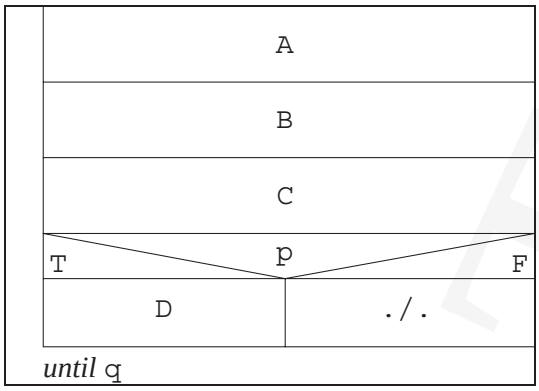


Abbildung A.3: Until-Iteration mit Alternative — als Struktogramm

A.1.21 Übung 3.3.5: Struktogramm⇒Java

Übung 3.3.5.1

Die Abbildung A.3 auf Seite 248 zeigt das Struktogramm.

Übung 3.3.5.2

```
 1  /**
 2   * Beispiel:
 3   * "Nichtabweisende Schleife" mit Alternative
 4   * Bonin 20-Nov-1999 08-Jun-2004
 5   */
 6  public class Ablauf {
 7      public static void main(String argv[]) {
 8          boolean p = true;
 9          boolean q = true;
10          do {
11              System.out.println("Run A");
12              System.out.println("Run B");
13              System.out.println("Run C");
14              if (p) System.out.println("Run D");
15          }
16      while (!q);
17      }
18  }
```

```
19  // End of object: Ablauf.java
20
```

Protokoll der Compilierung und Anwendung:

```
D:\bonin\widata\code>java -version
java version "1.4.2_03"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2_03-b02)
Java HotSpot(TM) Client VM
  (build 1.4.2_03-b02, mixed mode)

D:\bonin\widata\code>javac Ablauf.java

D:\bonin\widata\code>java Ablauf
Run A
Run B
Run C
Run D

D:\bonin\widata\code>
```

A.1.22 Übung 3.3.6: Text⇒ET-Verbundsystem

Tabelle A.1 Seite 250 zeigt die Lösung. Die Verknüpfung der einzelnen Entscheidungstabellen erfolgt hier über das Kommando `run`. Mit dem Kommando `return` wird Rücksprung zur aufrufenden ET verdeutlicht.

A.1.23 Übung 3.3.7: `Buchung` analysieren

Übung 3.3.7.1

- **ET-Buchung**: begrenzte Eintrefferentscheidungstabelle
- **ET-Saldierung**: begrenzte, komplexe Eintrefferentscheidungstabelle
- **ET-Nachforschung**: begrenzte Eintrefferentscheidungstabelle

Übung 3.3.7.2

- **ET-Buchung**: redundanzfrei, da keine gleichen Regeln
  2 Bedinungen $\rightarrow 2^2$ Regeln $= 4$; formal vollständig!

ET-WERBEAKTION		R1	R2	R3
B1	Alter < 14 Jahre ?	J	N	N
B2	Mitglied einer Gruppe?	-	J	N
A1	50% Rabatt	X		
A2	`run` ET-GRUPPE		X	
A3	`run` ET-EINZEL			X

ET-GRUPPE		R1	R2	R3	R4
B1	Gruppe ≥ 7 Mitglieder ?	J	J	J	N
B2	Kaufdatum 30 Werktage vorher?	J	N	N	-
B3	Kaufdatum 3 Werktage vorher?	-	J	N	-
A1	10% Rabatt	X			
A2	8% Rabatt		X		
A3	Kein Rabatt			X	
A4	`run` ET-EINZEL				X
A5	`return`	X	X	X	X

ET-EINZEL		R1	R2
B1	Kaufdatum 30 Werktage vorher?	J	N
A1	5% Rabatt	X	
A2	Kein Rabatt		X
A3	`return`	X	X

Tabelle A.1: ET-Verbundsystem: „Werbeaktion“

- **ET-Saldierung**: redundanzfrei, da keine gleichen Regeln
  3 Bedinungen $\rightarrow 2^3$ Regeln $= 8$ Regeln
  R3 repräsentiert 2 Regeln
  R4 repräsentiert 4 Regeln
  R1, R2 $= 2$ Regeln
  $\sum 8$ Regeln
  $\rightarrow$ formal vollständig!

- **ET-Nachforschung**: redundanzfrei, da keine gleichen Regeln
  1 Bedinungen $\rightarrow 2^1$ Regeln $= 2$; formal vollständig!

Übung 3.3.7.3

- **ET-Buchung**: R1 + R2 haben die gleichen Aktionsanzeigerfolgen und unterscheiden sich nur in B2 $\rightarrow$ konsolidierbar!
  R' mit B2 = – (Irrelevantzeichen) ersetzt R1 + R2

- **ET-Saldierung**: R1 + R2 haben die gleichen Aktionsanzeigerfolgen und unterscheiden sich nur in B3 $\rightarrow$ konsolidierbar!
  R' mit B3 = – (Irrelevantzeichen) ersetzt R1 + R2.

  Da B3 nun in allen Regeln für die Regelauswahl irrelevant ist, kann die Bedingung B3 ganz entfallen. Die ET hann dann nur noch 2 Bedingungen. Die Konsolidierung kann daher Einfluß auf die Zahl der Bedingungen haben.

- **ET-Nachforschung**: Die Aktionsanzeigerfolgen sind verschieden daher ist die ET nicht konsolidierbar!

A.1.24 Übung 3.3.8: `Workflow` analysieren

Übung 3.3.8.1

- **ET-Workflow**: begrenzte Eintrefferentscheidungstabelle
- **ET-Bearbeitung**: begrenzte, komplexe Eintrefferentscheidungstabelle
- **ET-Überprüfung**: begrenzte Eintrefferentscheidungstabelle

Übung 3.3.8.2

- **ET-Workflow**: redundanzfrei, da keine gleichen Regeln
  3 Bedinungen $\rightarrow 2^3$ Regeln $= 8$; formal vollständig!

- **ET-Bearbeitung**: redundanzfrei, da keine gleichen Regeln
  2 Bedinungen $\rightarrow 2^2$ Regeln $= 4$ Regeln
  R1 repräsentiert 2 Regeln
  R2 repräsentiert 1 Regeln
  $\sum 3$ Regeln
  $\rightarrow$ formal nicht vollständig!
  Es fehlt die Regel mit B1=N und B2=J. Eine zweckmäßige Aktionsanzeigerfolge ist A4=X

- **ET-Überprüfung**: redundanzfrei, da keine gleichen Regeln
  1 Bedinungen $\rightarrow 2^1$ Regeln $= 2$ Regeln; formal vollständig!

Übung 3.3.8.3

- **ET-Workflow**:
  - R5 + R7 haben die gleichen Aktionsanzeigerfolgen und unterscheiden sich nur in B2 $\rightarrow$ konsolidierbar!
    R5' mit B2 = – (Irrelevantzeichen) ersetzt R5 + R7
  - R6 + R8 haben die gleichen Aktionsanzeigerfolgen und unterscheiden sich nur in B2 $\rightarrow$ konsolidierbar!
    R6' mit B2 = – (Irrelevantzeichen) ersetzt R6 + R8

- **ET-Bearbeitung**: keine gleichen Aktionsanzeigerfolgen; nicht konsolidierbar!

- **ET-Überprüfung**: keine gleichen Aktionsanzeigerfolgen; nicht konsolidierbar!

Übung 3.3.9.4

Ja, wenn alle Anzeiger einer Bedingung irrelevant sind, dann ist diese Bedingung für die Auswahl keiner Regel bedeutsam. Diese Bedingung kann aus der Entscheidungstabelle entfernt werden.

A.1.25 Übung 3.3.10: Script `work.ksh`

Übung 3.3.10.1

```
$work.ksh konstrukte.txt ergebnis.txt↩
```

```
<body>
<h1>
<h2>
<h3>
<html>
<ol>
<p>
<title>
<ul>
```

Übung 3.3.10.2

```
Die Datei ergebnis.txt hat 9 Zeilen.
```

A.1.26 Übung 4.4.1: SOLL-Notation

Eine Anforderungsbeschreibung soll eindeutig, vollständig, verifizierbar, konsistent, modifizierbar, zurückführbar (*traceable*) und handhabbar sein.

A.1.27 Übung 4.4.2: UML-Modell

Übung 4.4.2.1

Die Abbildung A.4 S. 255 zeigt den Hauptteil des Klassendiagrammes für die SVI. In Abbildung A.5 S. 256 ist der Aspekt „Zielfernrohr“ abgebildet.

Übung 4.4.2.2

Die Abbildung A.6 S. 256 zeigt die Diagrammergänzung um den Aspekt „Waffenbesitzkarte“.

A.1.28 Übung 4.4.3: CGI versus SSI

CGI-Nachteil Im Normalfall verursacht CGI eine höhere Serverlast als SSI.

CGI-Vorteil CGI wird von (fast) allen Web-Servern verstanden. CGI ist daher als portabel zu betrachten. SSI ist weitgehend Server spezifisch.

A.1.29 Übung 4.4.4: Java & UML

Übung 4.4.4.1

↪Abbildung A.7 S. 257.

Übung 4.4.4.2

`28-Jun-2004`

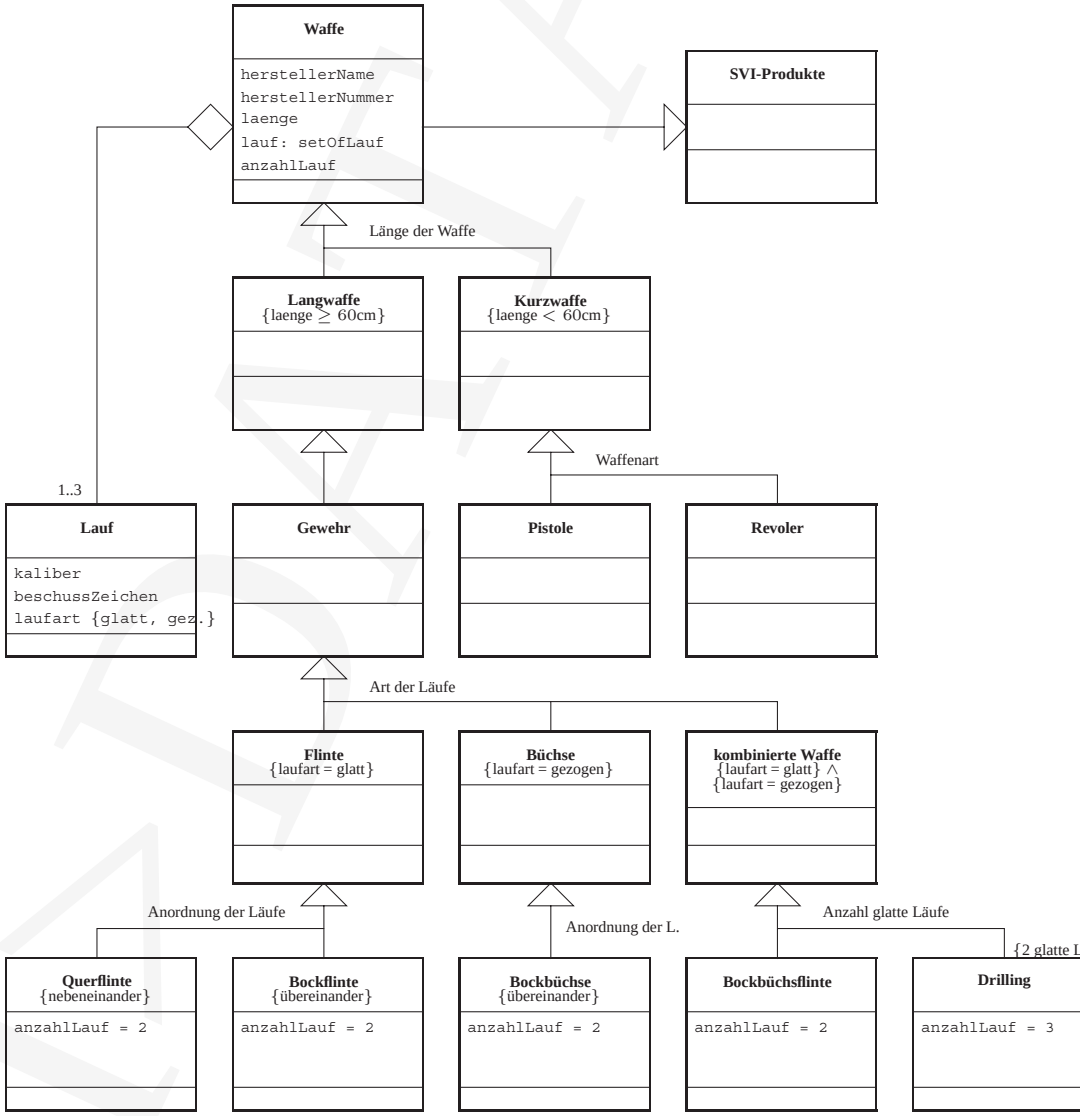


Abbildung A.4: Übung 4.4.2.1: SVI-Klassendiagramm „Hauptteil“

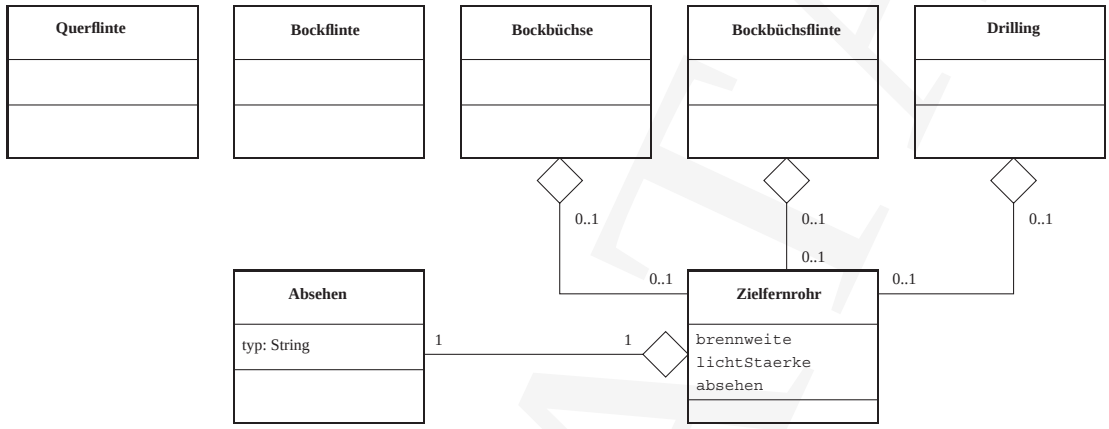


Abbildung A.5: Übung 4.4.2.1: SVI-Klassendiagramm „Zielfernrohr“

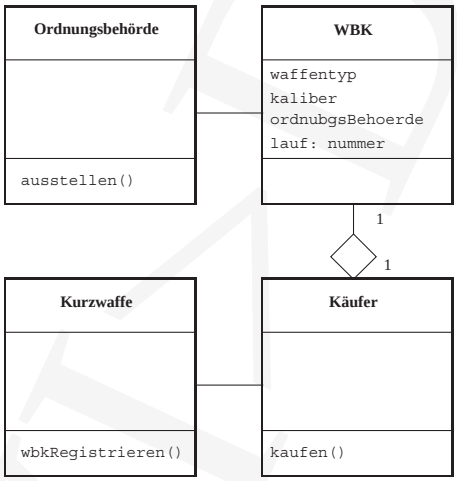
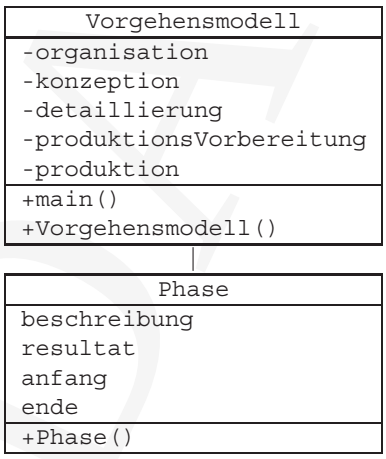


Abbildung A.6: Übung 4.4.2.2: SVI-Klassendiagramm „Waffenbesitzkarte“



Legende: Diagramm in UML-Notation

Abbildung A.7: Klassendiagramm: `Vorgehensmodell` & `Phase`

Station	Name	IP-Adresse
1	cisco-Lueneburg-32.grz.fh-lueneburg.de	193.174.32.19
2	FH-Lueneburg2.WiN-IP.DFN.DE	188.1.4.125
3	ZR-Hamburg1.WiN-IP.DFN.DE	188.1.144.169
4	ZR-Hannover1.WiN-IP.DFN.DE	188.1.144.22
5	IR-New-York1.WiN-IP.DFN.DE	188.1.144.138
6	IR-Perryman1.WiN-IP.DFN.DE	188.1.144.198
7	bordercore3-hssi0-0-0.Washington.cw.net	166.48.41.249
8	bordercore5.NorthRoyalton.cw.net	166.48.164.1
9	volkswagen-of-america.NorthRoyalton.cw.net	166.48.166.26

Legende: Ermittelt mit dem Kommando `traceroute`

Tabelle A.2: Stationen auf dem Weg von `as.fh-lueneburg.de` nach `www.audi-usa.com`

A.2 Tabellen

Als **Zahlenhierarchie** bezeichnet man die folgende Inklusionskette:

$$\mathbf{N} \subseteq \mathbf{Z} \subseteq \mathbb{Q} \subseteq \mathbf{R} \subseteq \mathbb{C}$$

mit:

Natürliche Zahlen†: $\mathbf{N} \equiv \{0, 1, 2, 3, \ldots\}$

Ganze Zahlen‡: $\mathbf{Z} \equiv \{\ldots, -2, -1, 0, 1, 2, \ldots\}$

Rationale Zahlen$^{\circ}$: $\mathbb{Q} \equiv \{\frac{m}{n}\}$ mit $m, n \in \mathbf{Z}$ und $n \neq 0$

Reelle Zahlen$^{\bullet}$: $\mathbf{R} \equiv$ Menge der reellen Zahlen, z. B.: π

Komlexe Zahlen$^{\diamond}$: $\mathbb{C} \equiv \{x + iy\}$ mit $x, y \in \mathbf{R}$

Legende:

Schreibweise:

- † Großbuchstabe „N“ mit einem doppelten Anfangsstrich
- ‡ Großbuchstabe „Z“ mit einem doppelten Diagonalstrich
- $^{\circ}$ Großbuchstabe „Q“ mit einem Strich
- $^{\bullet}$ Großbuchstabe „R“ mit doppeltem Anfangsstrich
- $^{\diamond}$ Großbuchstabe „C“ mit einem Strich

Tabelle A.3: Zahlenhierarchie

E	●	B	– ● ● ●	0	– – – – –
T	–	C	– ● – ●	1	● – – – –
A	● –	F	● ● – ●	2	● ● – – –
I	● ●	H	● ● ● ●	3	● ● ● – –
M	– –	J	● – – –	4	● ● ● ● –
N	– ●	L	● – ● ●	5	● ● ● ● ●
D	– ● ●	P	● – – ●	6	– ● ● ● ●
G	– – ●	Q	– – ● –	7	– – ● ● ●
K	– ● –	V	● ● ● –	8	– – – ● ●
O	– – –	X	– ● ● –	9	– – – – ●
R	● – ●	Y	– ● – –		
S	● ● ●	Z	– – ● ●		
U	● ● –				
W	● – –				

Legende:
Sortiert nach Längenklassen
Reihenfolge der Buchstabenhäufigkeit in englischen Texten:
{E,T,A,O,N,R,I,S,H,D,L,F,C,M,U,G,I,P,W,B,V,K,X,J,Q,Z}
Reihenfolge der Buchstabenhäufigkeit in deutschen Texten:
{E,N,R,I,S,T,D,H,A,U,L,C,G,M,O,B,Z,W,F,K,V, Ü,P,Ä,Ö,J,Y,Q,X}

Das Morsealphabet umfaßt im internationalen Telegraphieverkehr weitere „Sonderzeichen“; zum Beispiel das Komma oder den Doppelpunkt.

Tabelle A.4: Morse-Code (Ausschnitt)

Dezimal $B = 10$	**Hexadezimal**† $B = 16$	**BCD**‡ $B = 2$
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
–	A	1010
–	B	1011
–	C	1100
–	D	1101
–	E	1110
–	F	1111

Legende:

† $\equiv$ auch als „sedezimal“ bezeichnet

‡ $\equiv$ „duale Tetrade“ (Länge $= 4$ Bits)

B $\equiv$ Basis; $\hookrightarrow$Tabelle 2.4 auf Seite 81

Tabelle A.5: BCD (*Binary Coded Decimal*)

Hexa-dezi-mal	BCD 4-stel-lig	Dezimal 0	1	2	3	4	5	6	7
		BCD — 3-stellig							
		000	001	010	011	100	101	110	111
0	0000	NUL	DLE	Space	0	@	P	`	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	XON	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	XOF	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	′	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	-	=	M	]	m	}
E	1110	SO	RS	.	>	N	^	n	~
F	1111	SI	US	/	?	O	_	o	DEL

Legende:

Binary Coded Decimal (BCD)

American Standard Code for Information Interchange (ASCII)

ASCII-Code ≡ vordere drei Bits siehe Spalte plus hintere vier Bits siehe Zeile obiger Matrix — Beipiel: G ≡ 1000111.

Steuerzeichen ↪Tabelle A.7 auf Seite 263.

Tabelle A.6: ASCII-Code — ISO-7-Bit-Code —

NUL	alle Leitungen NULL	VT	Vertical Tab	SYN	Synchron. idle
SOH	Start Of Heading	FF	Form Feed	ETB	End Transmis. Block
STX	Start of TeXt	CR	Carriage Return	CAN	CANcel
ETX	End Of TeXt	SO	Shift Out	EM	End of Medium
EOT	End Of Transmission	SI	Shift In	SUB	SUBstitute
ENQ	ENQuiry	DLE	Data Link Escape	ESC	ESCape
ACK	ACKnowledged	DC1	Device Control 1	FS	File Separator
BEL	BELl	XON	XON-Protoc. (auch DC 2)	GS	Group Separator
BS	BackSpace	DC3	Device Control 3	RS	Record Separator
HT	Horizontal Tab	XOF	XOFF-Protoc. (auch DC 4)	US	Unit Separator
LF	Line Feed	NAK	Not AcKnowledgement	DEL	DELete

Legende:

↪Tabelle A.6 auf Seite 262

Tabelle A.7: Steuerzeichen im ASCII-Code

Bezeichnung	**Schreibweise**	**Sprechweise**	**Beispiel** $a \equiv 0011$ $b \equiv 0101$ **Wert:**
Konjunktion	$a \wedge b$	a und b	0001
Inhibition	$a \wedge \overline{b}$	a und nicht b	0010
	$\overline{a} \wedge b$	nicht a und b	0100
1. Projektion	a	a	0011
2. Projektion	b	b	0101
Negierte 1. Proj.	$\overline{a}$	nicht a	1100
Negierte 2. Proj.	$\overline{b}$	nicht b	1010
Disjunktion	$a \vee b$	a oder b	0111
Antivalenz (EXOR)	$a \oplus b$ $a \wedge \overline{b} \vee \overline{a} \wedge b$	entweder a oder b	0110
NOR-Funktion	$\neg(a \vee b)$ $\overline{a \vee b}$	weder a noch b	1000
Äquivalenz	$a \leftrightarrow b$ $a \wedge b \vee \overline{a} \wedge \overline{b}$	a genau dann, wenn b	1001
Implikation	$a \leftarrow b$	a gilt, wenn b gilt	1011
	$a \rightarrow b$	aus a folgt b	1101
	$\overline{a} \vee b$	wenn a, so b	
NAND-Funktion	$\neg(a \wedge b)$ $\overline{a \wedge b}$	nicht beide a und b	1110
Konstante 1	1	(stets) wahr	1111
Konstante 0	0	(stets) falsch	0000

Legende:
Ein *logischer Ausdruck* kann bestehen aus den Komponenten:
– logische Konstante 0, 1
– logische Variable (zum Beispiel a, b)
– Junktoren (zum Beispiel $\neg$, $\wedge$, $\vee$)
– Klammern *Hierarchieregel*: $\neg$ vor $\wedge$ vor $\vee$
Umformung ↪Tabelle A.9 auf Seite 265

Tabelle A.8: Junktoren

Bezeichnung	**Umformungsregel**
Doppelte Negation	$\overline{\overline{a}} \equiv a$
Kommuntativgesetz	$a \wedge b \equiv b \wedge a$ $a \vee b \equiv b \vee a$
Assoziativgesetz	$a \wedge (b \wedge c) \equiv (a \wedge b) \wedge c$ $a \vee (b \vee c) \equiv (a \vee b) \vee c$
Distributivgesetz	$a \wedge (b \vee c) \equiv (a \wedge b) \vee (a \wedge c)$ $a \vee (b \wedge c) \equiv (a \vee b) \wedge (a \vee c)$
Idempotenzgesetz	$a \wedge a \equiv a$ $a \vee a \equiv a$
Komplementgesetz	$a \wedge \overline{a} \equiv 0$ $a \vee \overline{a} \equiv 1$
01-Gesetze	$a \wedge 0 \equiv 0$ $a \wedge 1 \equiv a$ $a \vee 0 \equiv a$ $a \vee 1 \equiv 1$
Absorptionsgesetze	$a \wedge (a \vee b) \equiv a$ $a \vee (a \wedge b) \equiv a$ $(a \wedge b) \vee (a \wedge \overline{b}) \equiv a$ $(a \vee \overline{b}) \wedge b \equiv a \wedge b$ $(a \vee \overline{b}) \vee b \equiv a \vee b$ $(a \vee b) \wedge (a \vee \overline{b}) \equiv a$
Regel von *de Morgan*	$\overline{(a \wedge b)} \equiv \overline{a} \vee \overline{b}$ $\overline{(a \vee b)} \equiv \overline{a} \wedge \overline{b}$

Legende:
Junktoren ↪Tabelle A.8 auf Seite 264

Tabelle A.9: Boolsche Ausdrücke: Umformungsregeln

	Binärdarstellung		
Dezimal-darstel-lung	**Binärwert mit Vorzeichen**	**Einer-komplement** $K_1(x)$	**Zweier-komplement** $K_2(x)$
I	II	III	IV
-128	—	—	1000 0000
-127	1111 1111	1000 0000	1000 0001
-65	1100 0001	1011 1110	1011 1111
-15	1000 1111	1111 0000	1111 0001
-9	1000 1001	1111 0110	1111 0111
-1	1000 0001	1111 1110	1111 1111
-0	1000 0000	1111 1111	0000 0000
$+0$	0000 0000	0000 0000	0000 0000
$+1$	0000 0001	0000 0001	0000 0001
$+127$	0111 1111	0111 1111	0111 1111

Legende:
Wortlänge $n = 8$ Bit
$2^7 = 128_{10}$; $2^6 = 64_{10}$; $2^5 = 32_{10}$; $2^4 = 16_{10}$;
$2^3 = 8_{10}$; $2^2 = 4_{10}$; $2^1 = 2_{10}$; $2^0 = 1_{10}$

Tabelle A.10: Binäre Zahlenkomplemente

Lfd	**Typ**	**Menge**	**Einige Operationen**
1	`CARDINAL` `NATURAL`	$\mathbf{N} \equiv \{0, 1, 2, 3, \ldots\}$	Grundrechnungsarten: $+, -, *, /$, potenzieren Vergleiche: $<, \leq, =, \geq, >, \neq$ Vorgänger, Nachfolger, min, max für endliche Menge
2	`INTEGER`	$\mathbf{Z} \equiv \{\ldots, -1, 0, 1, \ldots\}$	`ABS` $(\|x\|)$ und `SGN` und wie bei 1
3	`REAL`	$\mathbb{Q} \equiv \{\frac{m}{n}\}$ mit $m, n \in \mathbf{Z}$ und $n \neq 0$	Wie bei 2, mit eingeschränktem Potenzieren, keine Vorgänger und Nachfolger
4	`BOOLEAN`	Wahrheitswerte $\{0, 1\}$	Junktoren $\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \oplus$ $\hookrightarrow$Tabelle A.8 auf Seite 264
5	`DOUBLE` `PRECISION`	$\mathbb{Q} \equiv \{\frac{m}{n}\}$ mit $m, n \in \mathbf{Z}$ und $n \neq 0$	Wie bei 3 mit doppelter Genauigkeit
6	`COMPLEX`	$\mathbb{C} \equiv \{x + iy\}$ mit $x, y \in \mathbf{R}$	Grundrechenarten mit Real- und Imaginärteil
7	`SET`	beliebige Menge	$\in, \notin, \subset, \subseteq, \cap, \cap, =, \neq$
8	`STRING`	beliebige Zeichenkette	Zeichenkettenoperationen

Tabelle A.11: Einfache Datenstrukturen

```
1  <!--================== HTML content models ================================-->
2
3  <!--
4      HTML has two basic content models:
5
6          %inline;     character level elements and text strings
7          %block;      block-like elements e.g. paragraphs and lists
8  -->
9
10 <!ENTITY % block
11      "P | %heading; | %list; | %preformatted; | DL | DIV | NOSCRIPT |
12       BLOCKQUOTE | FORM | HR | TABLE | FIELDSET | ADDRESS">
13
14 <!ENTITY % flow "%block; | %inline;">
15
16 <!--=================== Document Body =====================================-->
17
18 <!ELEMENT BODY O O (%block;|SCRIPT)+ +(INS|DEL) -- document body -->
19 <!ATTLIST BODY
20   %attrs;                              -- %coreattrs, %i18n, %events --
21   onload          %Script;   #IMPLIED  -- the document has been loaded --
22   onunload        %Script;   #IMPLIED  -- the document has been removed --
23   >
24
25 <!ELEMENT ADDRESS - - (%inline;)* -- information on author -->
26 <!ATTLIST ADDRESS
27   %attrs;                              -- %coreattrs, %i18n, %events --
28   >
29
30 <!ELEMENT DIV - - (%flow;)*           -- generic language/style container -->
31 <!ATTLIST DIV
32   %attrs;                              -- %coreattrs, %i18n, %events --
33   %reserved;                           -- reserved for possible future use --
34   >
35
```

Legende:

Quelle: `http://www.w3c.org/TR/Rec-html40/strict.dtd` Zugriff 01-Oct-1999

Hinweis: Bei der Elementdefinition wird *Tag Omission* (`OMITTAG`) zur Minimisierung des Schreibaufwandes genutzt. Das zweite „O“ (zum Beispiel bei `BODY` ↪Zeile 18) bedeutet, daß die Endemarke entfallen kann, weil aus dem Kontext darauf geschlossen werden kann. Bei einem Minuszeichen an dieser Stelle ist die Endmarke erforderlich. Das erste „O“ bezieht sich analog auf die Anfangsmarke.

Tabelle A.12: Auszug aus der *HTML 4.0 DTD* `strict.dtd`

Sprache	`lang=` wert
Arabisch	ar
Chinesisch	zh
Deutsch	de
Englisch	en
Französich	fr
Italenisch	it
Japanisch	ja
Niederländisch	nl
Griechisch	el
Spanisch	es
Portugisisch	pt
Russisch	ru

Legende: Zwei Buchstabenabkürzung für die Sprache gemäß ISO639:1988 ↪`http://www.oasis-open.org/cover/iso639a.html` Zugriff 07-Dec-1999

Tabelle A.13: Einige Werte für das Attribut `lang`

Lfd.	**Anforderungs-Kategorie**	**Kernfrage**	**Beispielformulierungen** (Programm DIAGNOSE)
1	funktionale Anforderungen	Was soll das System tun?	Aussagen sind aus Regeln abgeleitet oder werden beim Benutzer nachgefragt.
2	Anforderungen an das fertige Produkt	Was ist das System?	Das Dialogsystem DIAGNOSE ist ein Produkt für den Einsatz bei mehr als 100 Kfz-Betrieben.
3	Anforderungen zur System-umgebung	Was sind die Einsatzbe-dingungen?	DIAGNOSE setzt das Be-triebssystem Windows NT Version 4.0 voraus.
4	Anforderungen an die Ziel-struktur	Was ist der Bewertungs-hintergrund?	DIAGNOSE verkürzt die Fehlersuchzeit zumindest bei angelernten Kräften.
5	Anforderungen zur Projekt-durchführung	Was sind die Ressourcen für das Projekt-management?	DIAGNOSE Version 1.0 ist innerhalb von 3 Monaten mit 2 Personen zu entwickeln.

Tabelle A.14: Kategorien von Anforderungen: Programm DIAGNOSE

<!--#element attribute="value"-->			
element	attribute	value	Beschreibung
fsize	file	file-name	Ermittelt die Filegröße (Dimension siehe config.)
config	sizefmt	bytes	Filegröße wird in Bytes ermittelt.
		abbrev	Filegröße wird in Kilo- oder Mega-Bytes ermittelt.
	errmsg	text	Im Falle eines Server-Fehlers beim Parsen wird text gesendet.
echo	var	DOCUMENT_URI	Ermittelt den *Uniform Resource Identifier* (URI).
		LAST_MODIFIED	Ermittelt das Datum der letzten Änderung des Dokumentes.
		DATE_LOCAL	Ermittelt das lokale Datum des Web-Servers.
exec	cmd	shell-befehl	Fügt das Ergebnis des Shell-Kommandos ein.
	cgi	skript-name	Fügt das Ergebnis des Skripts ein.

Beispiel: Datum der letzten Dokumentänderung

```
<!--#echo var="LAST_MODIFIED"-->
```

Tabelle A.15: *Server-Side Includes*: einige *shtml*-Konstrukte

JavaScript Event Handler	
Event Handler	**Ereignis tritt ein, wenn der Benutzer** . . .
`onBlur`	Input beim Formularelement entfernt.
`onClick`	auf Link oder Formularelement klickt.
`onChange`	Text oder selektiertes Element ändert.
`onFocus`	ein Formularelement den Eingabefokus gibt.
`onLoad`	das Dokument lädt.
`onMouseOver`	den Mauszeiger über *Link* oder *Anker* bewegt.
`onSelect`	ein Formulareingabefeld selektiert.
`onSubmit`	ein Formular abschickt.
`onUnload`	das Dokument verläßt.

Tabelle A.16: *JavaScript* Event Handler

A.3 Abkürzungen und Akronyme

AH Application Hosting
AI Artificial Intelligence
AIX Advanced Interactive Executive
ALGOL Algorithmic Language
AOP Aspect-oriented Programming
API Application Programming Interface
ASCII American Standard Code for Information Interchange
ASP Active Server Pages
ASP Application Service Providing
ATM Asynchronus Transfer Modus
BASIC Beginners All Purpose Symbolic Instruction Code
BCD Binary Coded Decimal
BNF Backus-Naur Form
BPO Business Process Outsourcing
CASE Computer Aided Software Engineering
CBSE Component-Based Software Engineering
CCG Centrale für Coorganisation GmbH, Moorweg 133, D-50825 Köln
CD Compact Disc
CD-ROM Read Only Memory CD
CGI Common Gateway Interface
CLOS Common Lisp Object System
CMS Content Management System
COBOL Common Business-oriented Language
CPU Central Processing Unit
CSS Cascading Style Sheets
DBMS Datenbankmanagementsystem
DC Dublin Core Metadaten Initiative
DFÜ Datenfernübertragung
DHTML Dynamic HTML
DNS Domain Name Server
DOI Digital Object Identifier
DSS Decision Support System
DTD Document Type Definition

DV	Datenverarbeitung
DVD	Digital Versatile Disc
EAI	Enterprise Application Integration
EAN	europaeinheitliche Artikelnummer
EBCDIC	Extended Binary Coded Decimal Interchange Code
ECMA	European Computer Manufactures Association
ESIS	Element Structure Information Set
EVA	Eingabe Verarbeitung Ausgabe
ET	Entscheidungstabelle
FASMI	Fast Analysis of Shared Multidimensional Information
FINAL	Fachhochschule Nordostniedersachsen, Informatik, Arbeitsberichte, Lüneburg
FORTRAN	Formular Translation
FTP	File Transfer Program — oder Protocol)
GPRS	General Packet Radio System
GSM	Global System for Mobile Communication
GUI	graphical user interface tool
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
INRIA	Institut National de Recherche en Informatique et en Automatique
IP	Internet Protocol
IPO	Input Process Output
IPR	Intellectual Property Rights
ISBN	International Standard Book Number
ISO	International Standardization Organisation
JVM	Java Virtual Maschine
KSh	Korn-Shell
LAN	Local Area Network
LISP	List Processing Language
LOC	Lines of Code
MIME	Multipurpose Internet Mail Extensions
MIT	Massachusetts Institute of Technology
MJ	Manpower-Jahren
NDS	Nassi/Shneiderman diagram
#PCDATA	Parsed Character Data
OCR	optical character recognition

OLAP	On-line Analytical Processing
OLTP	Online Transaction Processing
P2P	Peer-to-Peer Network
PAP	Programmablaufplan
PCMCIA	Personal Computer Memory Card International Association
PDA	Personal Digital Assistant
PDL	process design language
PGP	Pretty Good Privacy
PHP	ursprünglich Personal Home Page Tool; heute rekursives Akronym PHP Hypertext Preprocessor
PICS	Platform for Internet Content Selection
PID	Process Identification (Number)
PL/I	Programming Language I
RDF	Resource Description Framework
RFC	Request for Comments
RFID	Radiofrequenzidentifikation
RMI	Remote Method Invocation
SGML	Standard Generalized Markup Language
SQL	Structured Query Language
SSI	Server-Side Includes
TGN	Getty Thesaurus of Geographic Names
UCS	Universal Multiple-Octet Coded Character Set
UML	Unified Modeling Language
UMTS	Universal Mobile Telecommunication(s) System (Services)
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTF	UCS Transformation Format
WAN	Wide Area Network
WAP	Wireless Application Protocol
WCMS	Web-Content-Management-System
WLAN	Wreless Local Area Network
WML	Wireless Markup Language
W3C	World Wide Web Consortium
XML	Extensible Markup Language — Akronym hätte normalerweise EML (Extensible Markup Language) lauten müssen, aber XML „sounds cooler“.
XSL	Extensible Stylesheet Language

A.4 Quellen

{{{{{{{{{{{{{{{{{{{{{{{{{}}}}}}}}}}}}}}}}}}}}}}}}}
{{{{{{{{{{{{{{{{{{{{{{{{{}}}}}}}}}}}}}}}}}}}}}}}}}
{{{{{{{{{{{{{{{{{{{{{{{{{}}}}}}}}}}}}}}}}}}}}}}}}}
{{{{{{{{{{{{{{{{{{{{{{{{{}}}}}}}}}}}}}}}}}}}}}}}}}
{{{{{{{{{{{{{{{{{{{{{{{{{}}}}}}}}}}}}}}}}}}}}}}}}}
{{{{{{{{{{{{{{{{{{{{{{{{{}}}}}}}}}}}}}}}}}}}}}}}}}
{{{{{{{{{{{{{{{{{{{{{{{{{}}}}}}}}}}}}}}}}}}}}}}}}}

World-**W**ide-**W**eb-Server:

`http://as.fhnon.de`

Unter diesem WWW-Server werden weitere Informationen zur Thematik „Einführung in die Wirtschaftsinformatik“ angeboten.

{{{{{{{{{{{{{{{{{{{{{{{{{}}}}}}}}}}}}}}}}}}}}}}}}}
{{{{{{{{{{{{{{{{{{{{{{{{{}}}}}}}}}}}}}}}}}}}}}}}}}
{{{{{{{{{{{{{{{{{{{{{{{{{}}}}}}}}}}}}}}}}}}}}}}}}}
{{{{{{{{{{{{{{{{{{{{{{{{{}}}}}}}}}}}}}}}}}}}}}}}}}
{{{{{{{{{{{{{{{{{{{{{{{{{}}}}}}}}}}}}}}}}}}}}}}}}}
{{{{{{{{{{{{{{{{{{{{{{{{{}}}}}}}}}}}}}}}}}}}}}}}}}
{{{{{{{{{{{{{{{{{{{{{{{{{}}}}}}}}}}}}}}}}}}}}}}}}}

Literaturverzeichnis

[App+00] Hans-Jürgen Appelrath / Jochen Ludewig; Skriptum Informatik — eine konventionelle Einführung, Stuttgart Leipzig (B. G. Teubner), 2000, ISBN 3-519-42153-4 — siehe auch [Lu91] — {Hinweis: Eine Einführung auf der Basis der Programmiersprache Modula-2.}

[Ba74] Friedrich L. Bauer; Was heißt und was ist Informatik, in: IBM-Nachrichten Nr. 223, Dezember 1974. {Hinweis: Eine Diskussion zur Frage, ob die Informatik eine Ingenieurwissenschaft ist.}

[Bo+91] Morris I. Bolsky / David G. Korn; Die KornShell — Beschreibung und Referenzhandbuch zur Befehls- und Programmiersprache, München (Hanser & Prentice-Hall), 1991 (deutsche Übersetzung: Manfred Schumacher). {Hinweis: Deutsche Fassung des Buches vom Schöpfer der Korn-Shell.}

[Bo83] Hinrich Bonin; Neue Impusle für die Verwaltungsautomation?, in: Verwaltungsführung Organisation Personal (VOP), 2/1983, S. 68–74. {Hinweis: Diskussion des Paradigmas.}

[Bo88] Hinrich Bonin; Die Planung komplexer Vorhaben der Verwaltungsautomation, Heidelberg (R. v. Decker & C. F. Müller), 1988 (Schriftenreihe Verwaltungsinformatik; Bd. 3). {Hinweis: Eine Kritik an der Softwareentwicklung nach dem Phasenmodell.}

[Bo91a] Hinrich Bonin; Cooperative Production of Documents, in: [Tr91], pp. 39–55. {Hinweis: Gedanken zur *Workflow*-Unterstützung.}

[Bo91b] Hinrich E. G. Bonin; Software-Konstruktion mit LISP, Berlin New York (Walter de Gruyter), 1991. {Hinweis: Eine ausführliche Darstellung der objektorientierung Softwareentwicklung auf der Basis der vielfältigen Scheme-Optionen.}

[Bo92a] Hinrich E. G. Bonin; Arbeitstechniken für die Softwareentwicklung, (3. überarbeitete Auflage Februar 1994), FINAL, 2. Jahrgang Heft 2, 10. September 1992, [FINAL]. {Hinweis: Bezieht auch Petri-Netze ein.}

[Bo92b] Hinrich E. G. Bonin; Teamwork between Non-Equals — Check-in & Check-out model for Producing Documents in a Hierarchy, in: SIGOIS Bulletin, Volume 13, Number 3, December 1992, (ACM Press), pp. 18–27. {Hinweis: Ein Vorschlag zur Automation von Vorgängen (Akten).}

[Bo92c] Hinrich E. G. Bonin (Hrsg.); Verwaltungsinformatik — Konturen einer Disziplin, Mannheim u.a. (BI Wissenschaftsverlag), 1992. {Hinweis: Stellt die Disziplin „Verwaltungsinformatik“ auf der Basis der Ergebnisse einer entsprechenden Fachtagung dar.}

[Bo93] Hinrich E. G. Bonin; The Joy of Computer Science, — Skript zur Vorlesung EDV —, Unvollständige Vorabfassung (4. Auflage März 1995), FINAL, 3. Jahrgang Heft 5, 20. September 1993, [FINAL]. {Hinweis: Ein Vorlesungsskript.}

[Bo94] Hinrich E. G. Bonin; Groupware-Systeme: Eine Perspektive für die öffentliche Verwaltung, in: Verwaltungsführung / Organisation / Personal (VOP), Heft 3, 1994, S. 170–176. {Hinweis: Eine Einschätzung der Leistungen und Wirkungen verschiedener Systeme.}

[Bo96] Hinrich E. G. Bonin; <HTML>-Ratgeber — Multimediadokumente im World-Wide Web programmieren, München Wien (Carl Hanser Verlag), 1996. {Hinweis: War zum Herausgabezeitpunkt ein Buch für die damals aktuellen Web-Fragen. Es enthält viele praktische Vorschläge zur Erstellung und Pflege einer Web-Publikation.}

[Bo98] Hinrich E. G. Bonin; Der Java-Coach, FINAL, 8. Jahrgang Heft 1, Oktober 1998, [FINAL], aktuelle Fassung unter:
`http://as.fhnon.de/publikation/anwdall.pdf` {Hinweis: Schließt beispielsweise die Themen „objekt-orientierte Datenbank“ (Fast Object) und „interne Klassen“ mit ein.}

[Bo00] Hinrich E. G. Bonin; Der kleine XMLer — XML verstehen und anwenden! Programmieren mit SVG —, aktuelle Fassung unter:
`http://as.fhnon.de/publikation/xmlerall.pdf` {Hinweis: Erläutert XML-Konstrukte im Zusammenhang mit Handlungsempfehlungen zum systematischen Programmieren.}

[Bo02] Hinrich E. G. Bonin; Aspect-Oriented Software Development — A Little Guidance to Better Java Applications —, aktuelle Fassung unter:
`http://as.fhnon.de/publikation/aosdall.pdf` {Remark: AOSD has the great conceptual vision that we could provide independent specifications for each distinct *concern* and then *weave* them together to build a valid application.}

[Bo03] Hinrich E. G. Bonin; Ruby Coach — Scripting in Ruby —, aktuelle Fassung unter:
`http://as.fhnon.de/publikation/rubyall.pdf` {Remark: Ruby is the interpreted scripting language for quick and easy object-oriented programming.}

[Br75] Frederick Brooks, Jr.; The Mythical Man-Month, Reading Massaschusetts u. a. (Addison-Wesley) 1975. {Hinweis: Die klassische Kritik an der Planungsgröße „LOC“.}

[Br+02] Manfred Broy / Dieter Rombach; Software Engineering — Wurzeln, Stand und Perspektiven, in: Informatik Spekrum, Band 25, Heft 6, Dezember 2002,

S. 438–451. {Hinweis: „Informatik ist nicht gleich Software Engineering, auch Softwaretechnik ist nicht gleich Software Engineering!“.}

[Codd70] E. F. Codd; A relational model for large shared databanks, in: Communications of the ACM, June 1970, Vol. 13, No. 6, pp. 377–387. {Hinweis: Orginalarbeit für das relationale Datenmodell.}

[Coy88] Wolfgang Coy; Aufbau und Arbeitsweise von Rechenanlagen — Eine Einführung in Rechnerarchitektur und Rechnerorganisation für das Grundstudium der Informatik —, Braunschweig Wiesbaden (Vieweg & Sohn), 1988. {Hinweis: Ein fundiertes Lehrbuch.}

[Coy+92] Wolfgang Coy / Frieder Nake / Jörg-Martin Pflüger / Arno Rolf / Jürgen Seetzen / Dirk Siefkes / Reinhard Stransfeld (Hrsg.); Sichweisen der Informatik, Baunschweig, 1992. {Hinweis: Eine gesellschaftspolitisch geprägte Analyse und Perspektive der Informatik.}

[DIN66241] Deutsches Institut für Normung e.V.; Entscheidungstabellen, Berlin Köln (Beuth Verlag) 1979. DIN-Hompage `ttp://www.din.de/` Zugriff 29-Dec-1999; Beuth-Verlag-GmbH-Homepage `http://www.beuth.de/` Zugriff 29-Dec-1999; {Hinweis: Normierte Notation von Struktogrammen.}

[DIN66261] Deutsches Institut für Normung e.V.; Sinnbilder nach Nassi-Shneiderman und ihre Anwendung in Programmablaufplänen, Berlin Köln (Beuth Verlag) 1984. DIN-Hompage `ttp://www.din.de/` Zugriff 29-Dec-1999; Beuth-Verlag-GmbH-Homepage `http://www.beuth.de/` Zugriff 29-Dec-1999; {Hinweis: Normierte Notation von Struktogrammen.}

[DuCh1999] Bob DuCharme; XML — The Annotated Specification, (Prentice Hall PTR), ISBN 0-13-082676-6, 1999. {Hinweis: Eine Erläuterung der offiziellen XML-Empfehlung vom 10-Februar-1998.}

[Eck1990] Erhart Ecker; Einführung in die Informatik, Berlin Offenbach (VDE-Verlag), 1990. {Hinweis: Ein Lehrbuch primär für Elektroingenieure.}

[FINAL] Fachhochschule Nordostniedersachsen, Informatik, Arbeitsberichte, Lüneburg (FINAL) herausgegeben von Hinrich E. G. Bonin, ISSN 0939-8821, ab 7. Jahrgang (1997) auf CD-ROM, beziehbar: FH NON, Volgershall 1, D-21339 Lüneburg, Germany. {Hinweis: Eine Reihe, die primär von Lehrenden und Studierenden der FH Nordostniedersachsen getragen wird.}

[Fl85] Christiane Floyd; Wo sind die Grenzen des verantwortbaren Computereinsatzes?, in: Informatik-Spektrum, Band 8, Heft 1, 1985, S. 3–6. {Hinweis: Ein klassischer Definitionsansatz einer Ethik für Informatiker.}

[Fo+98] Martin Fowler / Kendall Scott; UML konzentriert — Die neue Standard-Objektmodellierungssprache anwenden, ISBN 3-8273-1329-5, (Addision-Wesley), 1998. {Hinweis: Eine kurze und prägnante UML-Einführung.}

[Go+99] Michel Goossens / Sebastian Rahtz / Eitan Gurari / Ross Moore / Robert Sutor; The LaTeXWeb Companion, Integrating TeX, HTML, and XML — Tools and Techniques for Computer Typesetting, ISBN 0-201-43311-7, (Addision-Wesley), 1999. {Hinweis: Ein hervorragendes Buch zur Frage wie in Zukunft ein Web-Dokument erzeugt werden kann. Befaßt sich zum Beispiel ausführlich mit DSSSL. CSS, XSL.}

[Han96] Hans Robert Hansen; Wirtschaftsinformatik I — Einführung in die betriebliche Datenverarbeitung, Stuttgart Jena (Gustav Fischer Verlag, UTB 802), 7., völlig neubearbeitete und stark erweiterte Auflage, 1996, ISBN 3-8252-0802-8. {Hinweis: In der jeweils aktuellen Auflage stets ein „Klassiker“.}

[Hei00] Marcus van der Heijden / Marcus Taylor (Eds.); Understanding WAP — Wireless Applications, Devices, and Services, Boston, London (Artech House), 2000, ISBN 1-58053-093-1. {Hinweis: Leicht verstehbare, einführende Artikel zur WAP-Technologie.}

[Hof83] Helmut Hofstetter; Organisationspsychologische Aspekte der Softwareentwicklung, in: Heinz Schelle / Peter Molzberger (Hrsg.); Psychologische Aspekte der Software-Entwicklung, München Wien (R. Oldenbourg Verlag) 1983, S. 25–62. {Hinweis: Kritische Auseinandersetzung mit einer technikbezogenen Sicht, die den Mensch vernachläßigt.}

[Hol+00] Harri Holma / Antti Toskala (Eds.); WCDMA for UMTS — Radio Access For Third Generation Mobile Communications, Chichester, New York (John Wiley), 2000, ISBN 0-471-72051-8. {Hinweis: Elektrotechnisch geprägte Artikel zu UMTS.}

[Ho99] Alex Homer; XML in IE5 — Programmer’s Reference, ISBN 1-861001-57-6, (WROX), 1999. {Hinweis: Ein gelungener XML-Ratgeber für Microsoft’s Internet Explorer 5.}

[Kl03] Helmut Klemm; Ein großes Elend — Das Informationszeitalter kann sich nicht einigen über den Begriff „Information“, in: Informatik Spekrum, Band 26, Heft 4, August 2003, S. 267–273. {Hinweis: Primär eine Fortführung der Begriffsdiskussion die von Günter Ropohl (Technikwissenschaftler, Universität Frankfurt) in der Zeitschrift „Ethik und Sozialwissenschaften“ angestossen wurde.}

[Kre03] Hans-Jörg Kreowski; Syntax, Sematik und . . ., in: [Rö03] S. 75–86.

[La+99] Simon St. Laurent / Robert Biggar; Inside XML DTDs, New York u. a. (McGraw-Hill), 1999. {Hinweis: Ein leicht verständliches und informatives Buch über DTDs in XML. Gibt Beispiele über spezielle Anwendungsfelder.}

[Li+79] Richard C. Linger / Harlan D. Mills / Bernard I. Witt; Structured Programming: Theory and Practice, Reading, Massachusetts (Addison-Wesley) 1979. {Hinweis: Das Werk über PDL (*Process Design Language*).}

[Lu91] Jochen Ludewig; Einführung in die Informatik — Skriptum Informatik I, II —, (3. durchgesehene Auflage 1991), Zürich (Verlag der Fachvereine Zürich), 1991. — siehe auch [App+00] — {Hinweis: Ein spannendes Lehrbuch, auch in der aktuellen Auflage.}

[Mau04] Hermann Maurer; Der PC in zehn Jahren, in Informatik-Spektrum, Band 27, Heft 1 2004, S. 44–50. {Hinweis: Eine Skizze des Nachfolgers des klassischen Personalcomputers.}

[Me+04] Peter Mertens / Reimut Bodendorf / Wolfgang König / Arnold Picot / Matthias Schumann /Thomas Hess; Grundzüge der Wirtschaftsinformatik, Berlin Heidelber u. a. (Springer) 1991, fünfte, neubearbeitete Auflage 1998, ISBN 3-540-63752-4; achte Auflage 2004, ISBN 3-540-40687-5. {Hinweis: Ein Lehrbuch, das integrierte Anwendungssysteme in den Mittelpunkt der Betrachtungen stellt.}

[Na92] Frieder Nake; Informatik und Maschinisierung von Kopfarbeit, in: [Coy+92], S. 181-201. {Hinweis: Zum Schaffen von Frieder Nake ↪ [Rö03].}

[Na03] Frieder Nake; Der Computer als Automat, Werkzeug und Medium und unser Verhältnis zu ihm, in: [Rö03], S. 212–223. {Hinweis: 14 Thesen erstmals vorgetragen beim Kolloquum „Menschenbild und Computer“ am 10/11-Jul-1999 in Bremen.}

[Nas+73] I. Nassi / B. Shneiderman; Flowchart Techniquies for Structured Programming; in: SIGPLAN Notices 8 (1973), pp. 12–26. {Hinweis: Orginalarbeit über Struktogramme.}

[Neu98] Horst A. Neumann; Objektorientierte Softwareentwicklung mit der Unified Modeling Language (UML), ISBN 3-446-18879-7, München Wien (Carl Hanser Verlag), 1998. {Hinweis: Eine umfassende Darstellung von Konzepten und Vorgehensweisen.}

[Rech91] Peter Rechenberg; Was ist Informatik?: Eine allgemeinverständliche Einführung, München Wien (Carl Hanser Verlag) 1991, ISBN 3-446-16324-7. {Hinweis: Ein Lehrbuch für „gebildete Informatik-Laien“ mit dem Anspruch die „ganze Informatik“ abzudecken.}

[Rech03] Peter Rechenberg; Zum Informationsbegriff der Informationstheorie, in Informatik-Spektrum, Band 26, Heft 5, 2003, S. 317–326. {Hinweis: Eine fundierte Erläuterung des Begriffs „Information“.}

[Rech04] Peter Rechenberg; Stellungnahme des Verfassers zur Diskussion des Begriffs Information, in Informatik-Spektrum, Band 27, Heft 1, 2004, S. 92–93. {Hinweis: Eine Zusammenfassung von Diskussionsbeiträgen über den Begriff „Information“.}

[Rö03] Karl-Heinz Rödiger (Hrsg.); Algorithmik — Kunst — Semiotik, Hommage für Frieder Nake, Heidelberg (Synchron Wissenschaftsverlag) 2003, ISBN 3-935025-60. {Hinweis: Festschrift für Frieder Nake, einer der großen Pioniere der Computergraphik.}

[St+99] Peter Stahlknecht / Ulrich Hasenkamp; Einführung in die Wirtschaftsinformatik, 9., vollständig überarbeitete Auflage, Berlin Heidelber u. a. (Springer) 1999, ISBN 3-540-65764-9. {Hinweis: Ein Lehrbuch, das eine praxisbezogene Einführung in alle Bereiche der WI vermittelt. In der jeweiligen Auflage eine umfassende Darstellung des Gesamtgebietes.}

[Str77] W. Strunz; Entscheidungstabellentechnik — Grundlagen und Anwendungsmöglichkeiten bei der Gestaltung rechnergestützter Informationssysteme, München Wien 1977. {Hinweis: Das klassische Werk über ET.}

[Tr91] Roland Traunmüller (Editor); Governmental and Municipal Information Systems, II, IFIP, Amsterdam, u. a. (North-Holland), 1991. {Hinweis: Berichte einer Fachtagung über die Verwaltungsinformatik.}

[Var+00] Upkar Varshney / Ron Vetter; Emerging Mobile and Wireless Networks, in: Communications of the ACM, June 2000, Vol. 43, No. 6, pp. 73–81. {Hinweis: Ein Trendartikel für leitungslose Netzwerke.}

[Zi93] Miklós G. Zilahi-Szabó; Wirtschaftinformatik — Anwendungsorientierte Einführung —, München ien (R. Oldenbourg Verlag), 1993. {Hinweis: Ein umfassendes Werk mit Nachschlagecharakter.}

A.5 Index

Index

Index

⋆ ⋆ ⋆