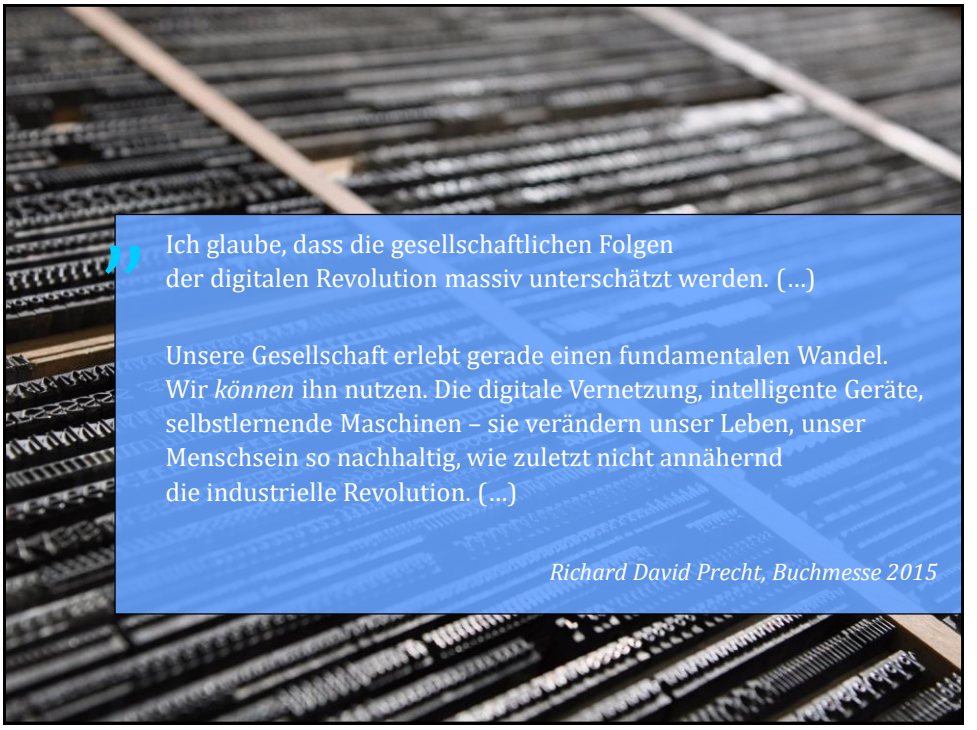
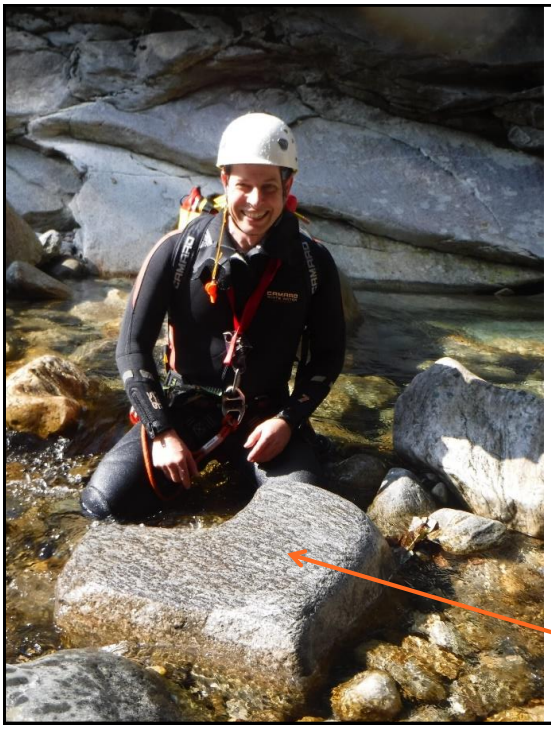

Ich glaube, dass die gesellschaftlichen Folgen
der digitalen Revolution massiv unterschätzt werden. (…)
Unsere Gesellschaft erlebt gerade einen fundamentalen Wandel.
Wir *können* ihn nutzen. Die digitale Vernetzung, intelligente Geräte,
selbstlernende Maschinen – sie verändern unser Leben, unser
Menschsein so nachhaltig, wie zuletzt nicht annähernd
die industrielle Revolution. (…)
Richard David Precht, Buchmesse 2015

80% der Menschen
sehen hier ein Herz.
2 / 123



Du siehst eine Spitzklammer?

Dann bist Du hier richtig!

Tobias Ott

seit 1996 Geschäftsführer von pagina GmbH Publikationstechnologien

1997-2006 Lehrbeauftragter für „Elektronisches Publizieren“
2006-2015 Vertretungsprofessor „Cross Media Publishing“
seit 2017 Lehrbeauftragter für „Digital Publishing“

Schwerpunkte bei pagina:
Strategische Verlagsberatung, Digitale Geschäftsmodelle, XML-Workflows

Forschungsschwerpunkte:
Die Rolle der Verlage im 21. Jahrhundert
Die Auswirkungen der Digitalisierung auf NGOs und ihre Rolle in der modernen Gesellschaft

Letzte Publikationen:
„Crossmediales Publizieren im Verlag“ (De Gruyter 2013)
„Print CSS. Einsatz und Referenz“ (Hrsg., pagina, 2015)

1973 „pagina GmbH Elektronische Satzherstellung“
1998 „pagina GmbH Gesamtherstellung wissenschaftlicher Werke“
2011 „pagina GmbH Publikationstechnologien“

pagina Publikations technologien

28 Mitarbeiter	(4 ProjektmanagerInnen, 9 EntwicklerInnen, 8 SetzerInnen, 2 Konvertierer, 3 Contentmanager)
5 Satztechnologien	(InDesign, Miles Oasys, TUSTEP, XSL-FO, PrintCSS)
5 MAM/CMS-Partnerschaften	(SiteFusion, Censhare, Sirius, Six, celum)
2 Technologie-Partnerschaften	(oXygen, SMASHdocs)
5+ Publishing-Lösungen	(parsX, OpenContent, SiteFusionX, Epub-Checker, DH-Lösungen)
(Fast) Alle XML-Technologien	(XSLT, XPath, XSD, DTD, RelaxNG, Schematron, XProc, eXist, xBase, XQuery)

Geschäftsbereiche 2018:
Medienproduktion Entwicklung Schulung & Beratung MAM-Systeme Digital Humanities

Derzeit verantwortlich für die XML-Workflows u.a. bei:
S. Fischer Rowohlt Droemer Knaur MairDumont Gräfe&Unzer Oetinger Hofmann&Campe
W. Kohlhammer J.C.B. Mohr Meyer&Meyer Thorbecke Motorpresse Duden Kampa
dtv BUND-Verlag Diogenes Ulmer Spotlight Narr/francke Kiepenheuer & Witsch
Verlag für Versicherungswirtschaft Sauerländer GOVI W1 Media Reclam IUBH

Agenda

XUGS, Mittwoch 7.11.2018

1. (wer braucht eigentlich)
   Verlage im 21. Jahrhundert
   Publizieren in der digitalen Gesellschaft
2. XML macht Sinn:
   der Grund
3. XML ist nicht schwer:
   die Technik
4. XML kann leicht kontrolliert werden:
   Die Grammatik(en)

Ein kleines

GEDANKENEXPERIMENT

zum Start:

Wenn man Verlage heute erfinden würde:

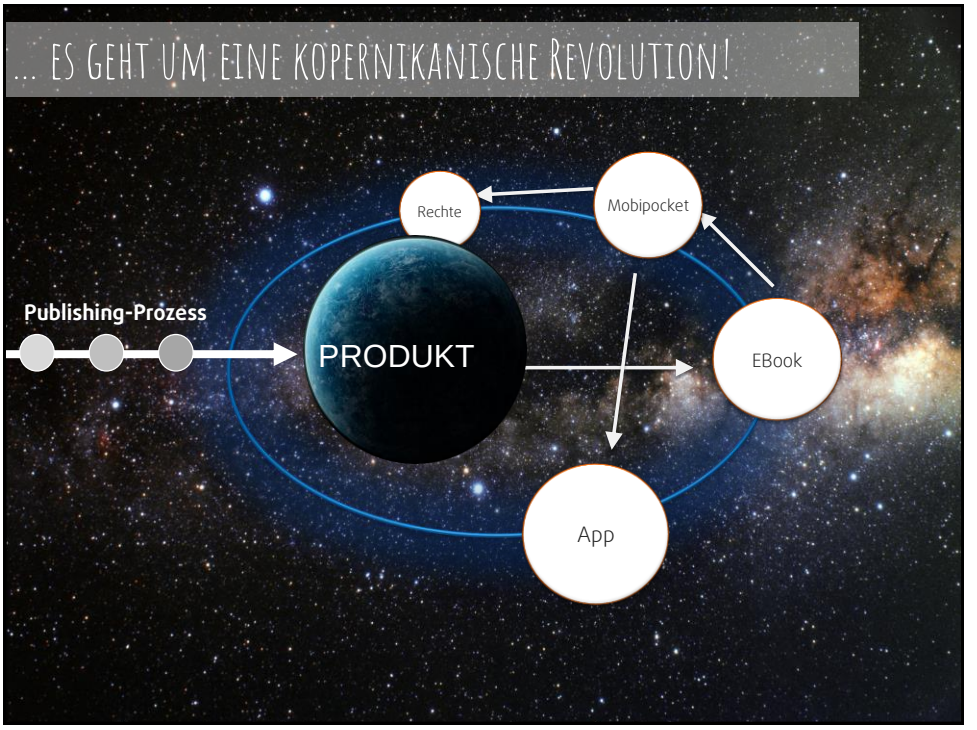
Wie würden sie aussehen?

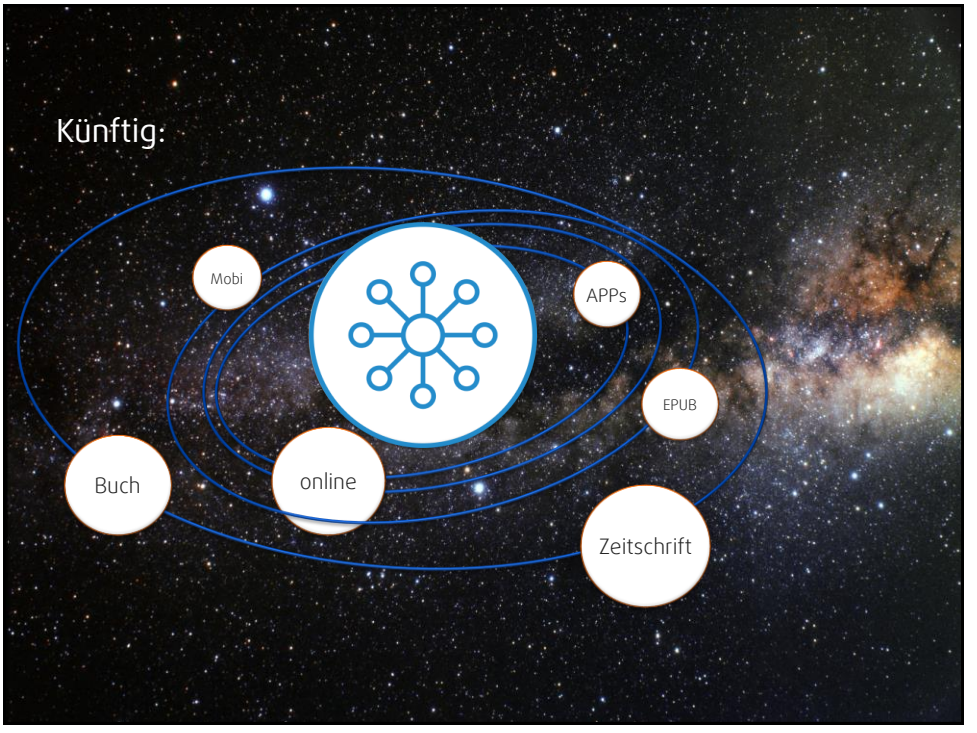
Mit welchem Geschäftsmodell würden wir starten?

Welche Produkte und Services würden wir anbieten?

Wie selbstverständlich wären wir Teil der digitalen Gesellschaft?

...WORUM GEHT ES HIER EIGENTLICH?


... es geht um eine kopernikanische Revolution!
Publishing-Prozess
PRODUKT
Rechte
Mobipocket
EBook
App


Künftig:
Mobi
APPs
EPUB
Buch
online
Zeitschrift

Was aber bedeutet das für den Verlag?

Verlag und Leser

TRENDS

Verlag als Informationsanbieter

Verlag als Servicegeber

Content Syndication

Direktgeschäft

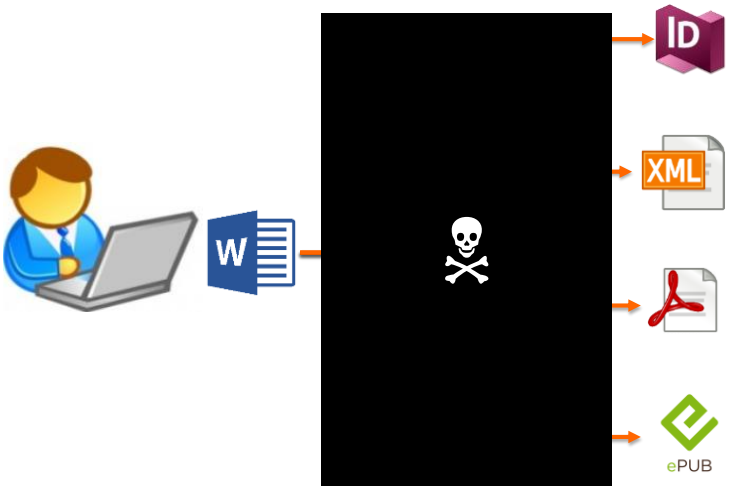
Individualisierung & Personalisierung

Crossmediales Publizieren

Digital first

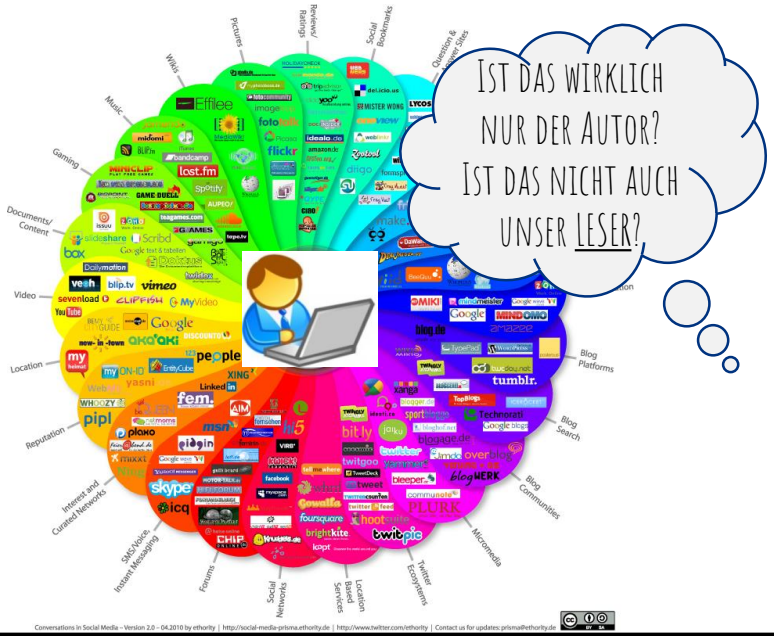
Open Access

Das Phänomen Autor



...wie Verlage ihn sehen

… wie ihn die digitale Gesellschaft (und vielleicht auch er sich selbst) sieht



Trends

Kollaboratives Arbeiten

Schreiben als Prozess (der niemals endet)

Rückspielen der Daten an den Autor

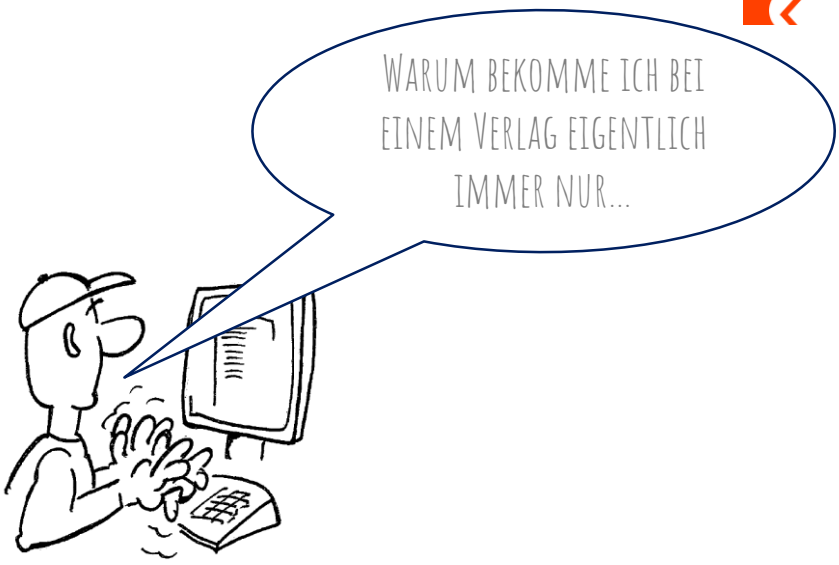
Leser wird zum Co-Autor

Communities

Social Media (Marketing)

Selfpublishing

…



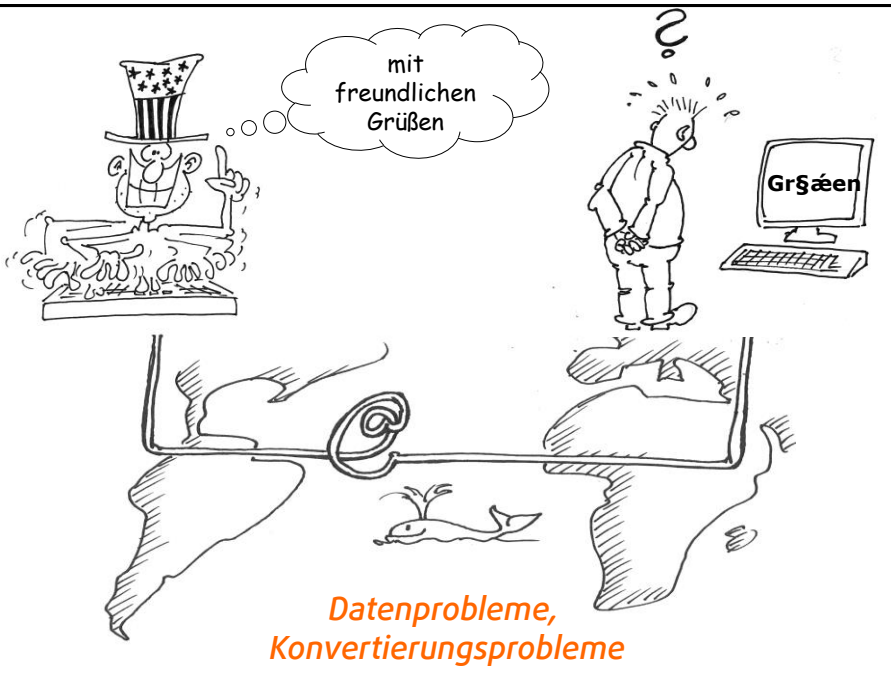
Nochmals: Was bedeutet das für den Verlag?

1. XML

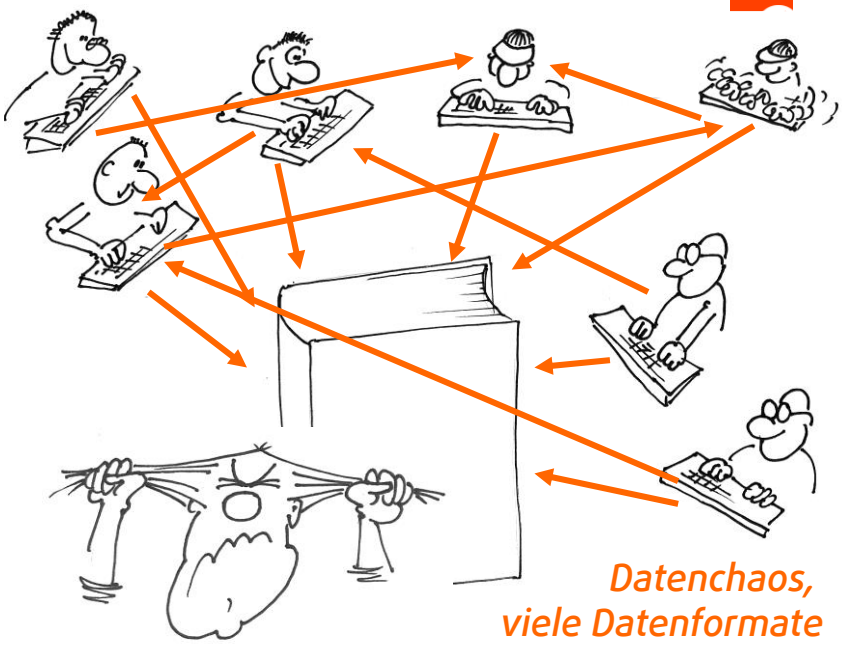
macht Sinn: Der Grund

warum etwas neues?

Es läuft doch super...

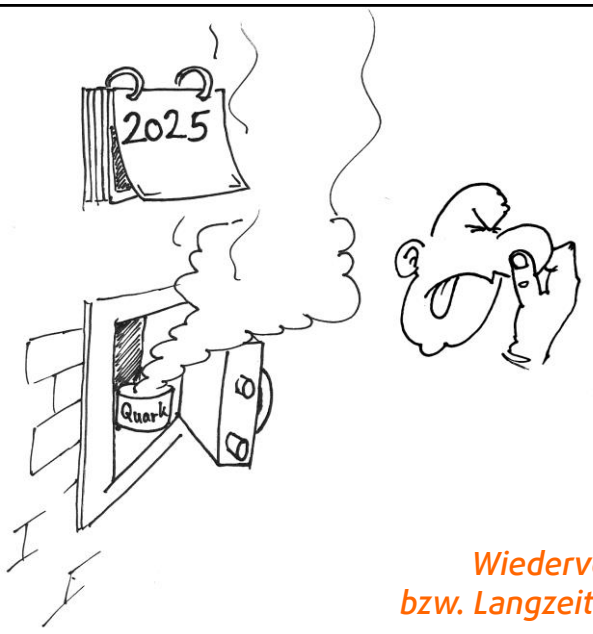


Datenprobleme, Konvertierungsprobleme

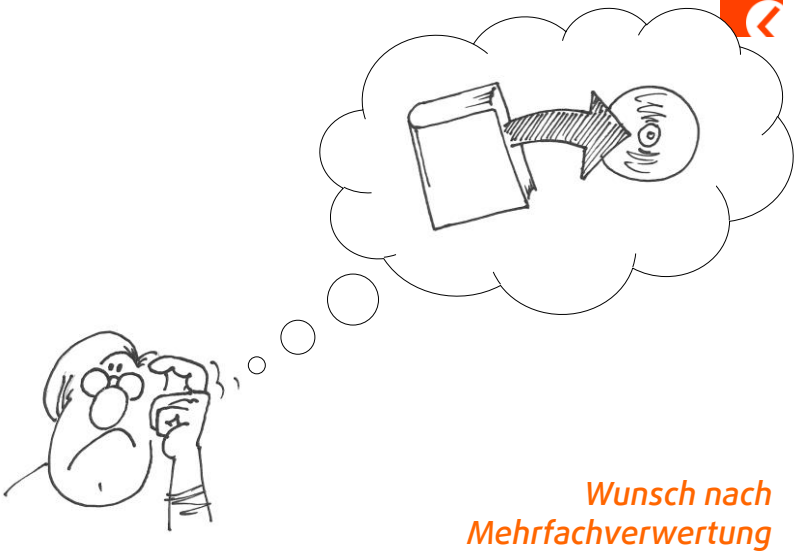


Datenchaos,
viele Datenformate

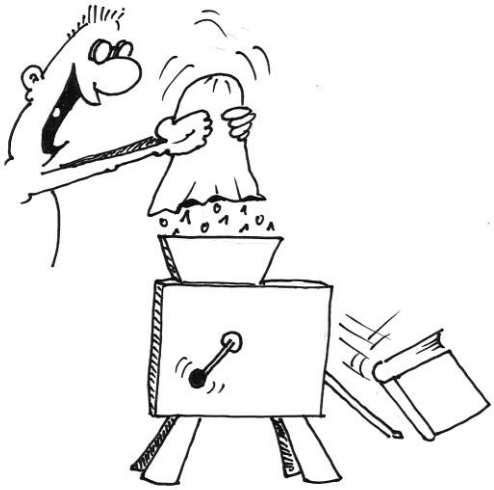
Hohe laufende
Kosten im Satz
auch bei
gleichartigen Publikationen



Keine Wiederverwertbarkeit bzw. Langzeitverfügbarkeit der Satzdaten



Wunsch nach Mehrfachverwertung der Daten



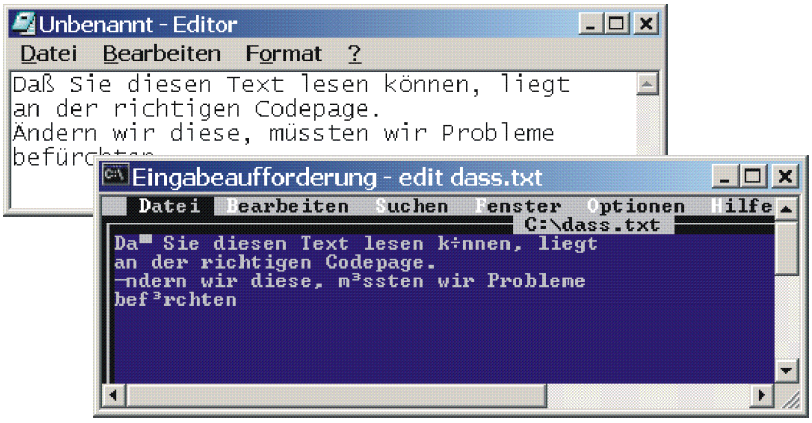
Wunsch nach automatisierter Produktion

2. XML

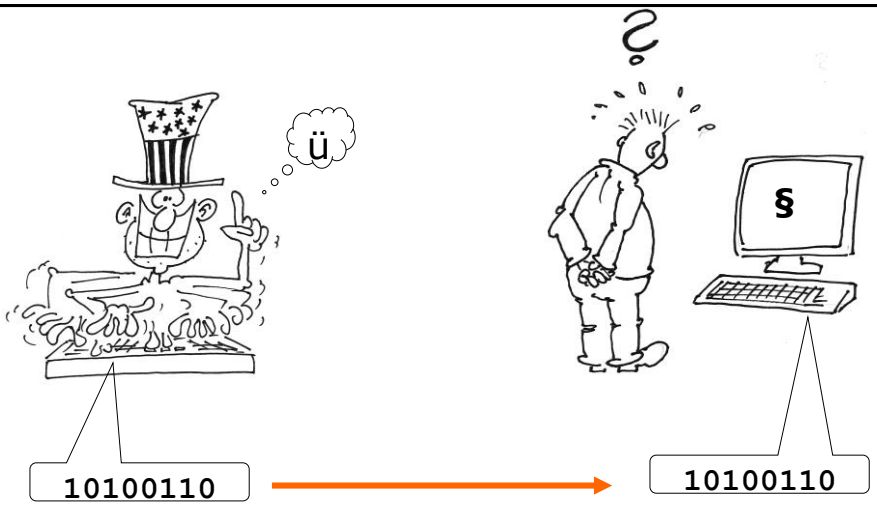
ist nicht schwer: die Technik

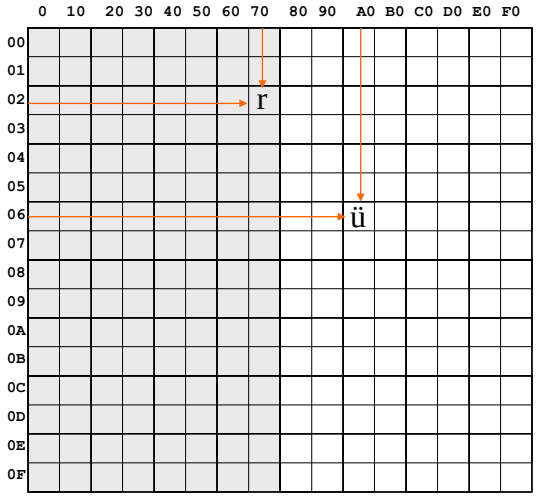
...Sie brauchen für den Einstieg nur sechs Begriffe!

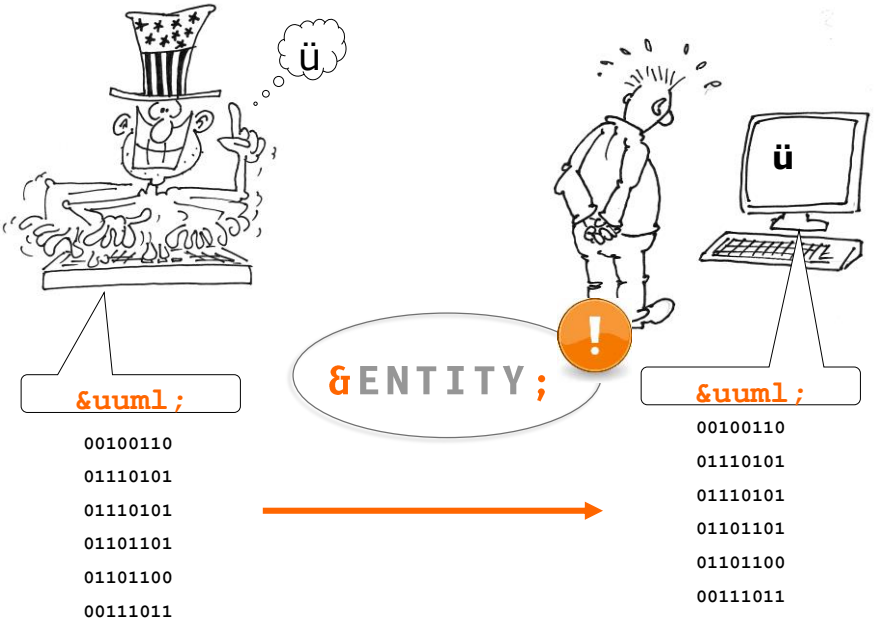
Das »Geheimnis der Zeichencodierung«

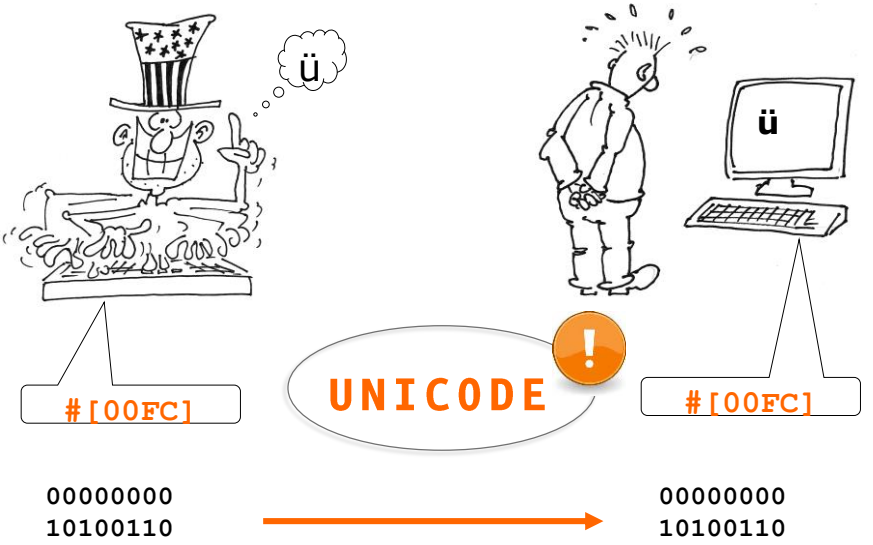


Erste Voraussetzung also:
Plattformneutralität




0 10 20 30 40 50 60 70 80 90 A0 B0 C0 D0 E0 F0
00
01
02
03
04
05
06
07
08
09
0A
0B
0C
0D
0E
0F
r
ü


ü
ü
ü
&ENTITY;
ü
00100110
01110101
01110101
01101101
01101100
00111011
00100110
01110101
01110101
01101101
01101100
00111011



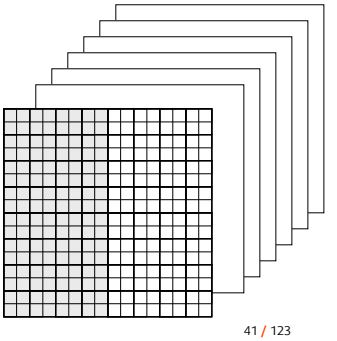
Lösung: UNICODE
statt individueller Codepages

UNICODE hat sich als neuer Standard der Zeichencodierung durchgesetzt.

Durch die Zuordnung von 2 Byte je Zeichen stehen nun im „Basic Mulitlingual Plane“ (BMP) 2^{16} = über 65.000 eindeutig zugewiesene Zeichen zur Verfügung.

Es gibt in Unicode insgesamt 17 solcher Planes (2-Byte-Bereiche), so dass in Wirklichkeit 17x65.000 Zeichen zur Verfügung stehen. Die meisten davon sind noch nicht definiert.

XML-Daten sind UNICODE-codierte Daten gemäß dem Codierungsschema UTF-8 *(8-bit Universal Character Set Transformation Format),* in dem die Daten auf 1-4 Byte pro Zeichen abgelegt werden, so dass theoretisch über 1 Mio Zeichen definiert werden können.



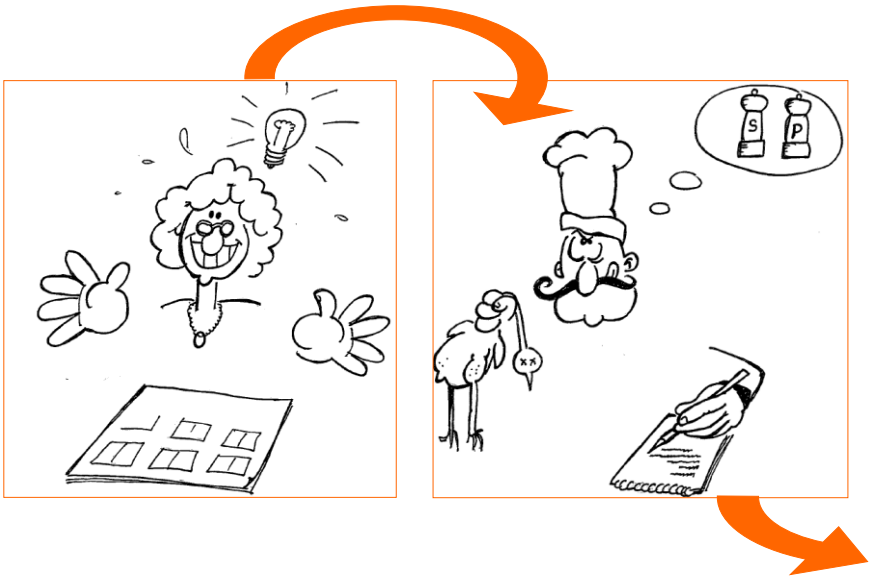
Interessante Links zum Thema UNICODE:

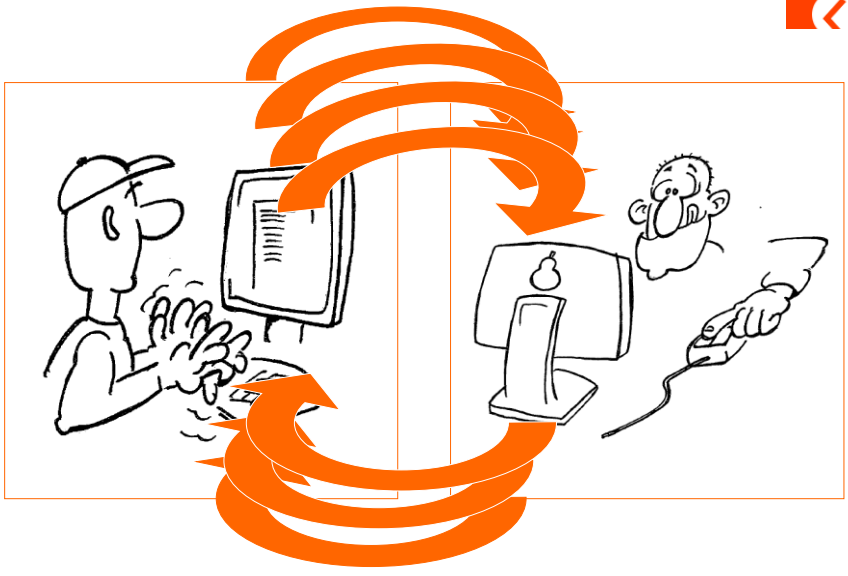
http://www.utf8-zeichentabelle.de/
http://www.unicode.org/charts/
http://www.decodeunicode.org/

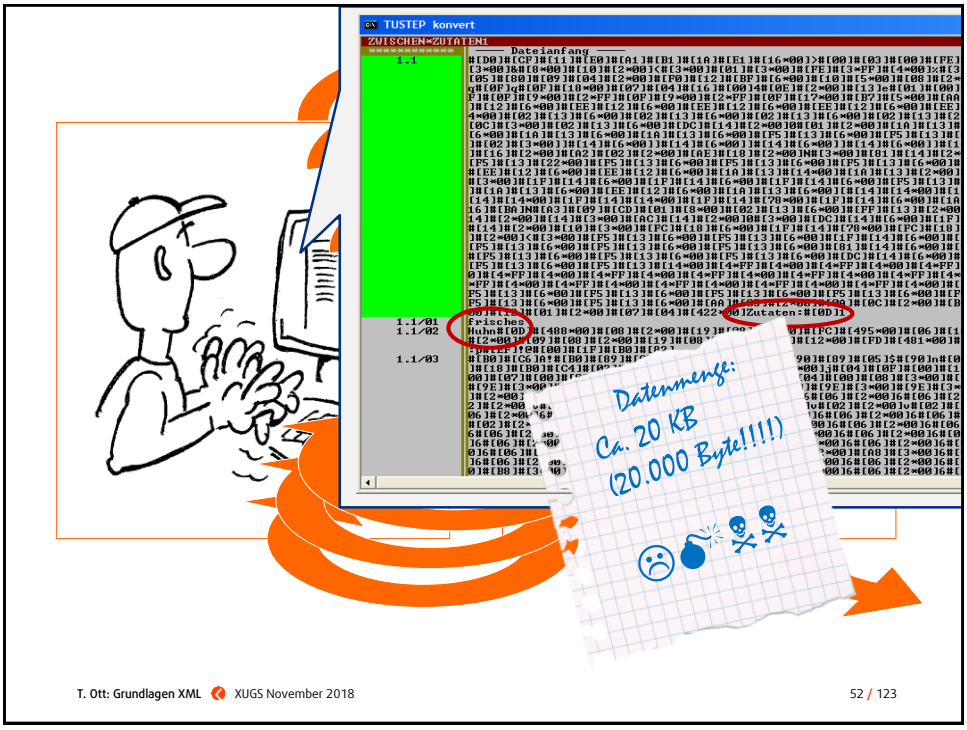
...doch die ZEICHEN-Codierung ist das geringste Problem...

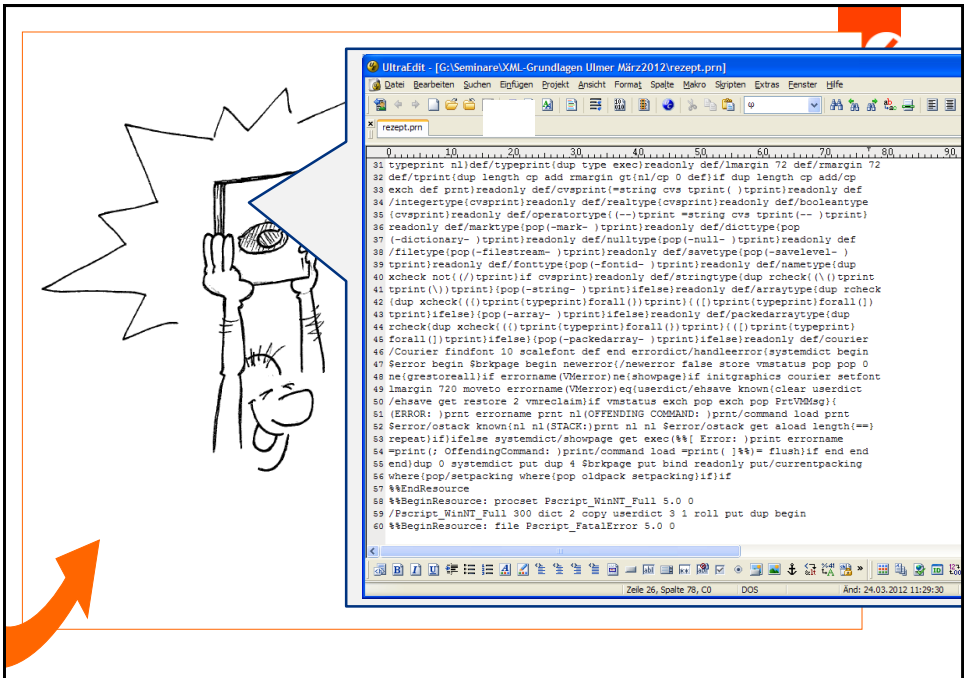
Zutaten:
1 frisches Huhn

Der konventionelle, printzentrierte Workflow (nicht so gut)






TUSTEP konvert
ZWISCHEN=ZUTATEN1
Dateianfang
Datenmenge:
Ca. 20 KB
(20.000 Byte!!!!)


UltraEdit - [G:\Seminare\XML-Grundlagen Ulmer März2012\rezept.prn]
Datei Bearbeiten Suchen Einfügen Projekt Ansicht Format Spalte Makro Skripten Extras Fenster Hilfe
rezept.prn
31 typeprint nl}def/typeprint{dup type exec}readonly def/lmargin 72 def/rmargin 72
32 def/tprint{dup length cp add rmargin gt{nl/cp 0 def}if dup length cp add/cp
33 exch def prnt}readonly def/cvsprint{=string cvs tprint( )tprint}readonly def
34 /integertype{cvsprint}readonly def/realtype{cvsprint}readonly def/booleantype
35 {cvsprint}readonly def/operatortype{(--)tprint =string cvs tprint(--)tprint}
36 readonly def/marktype{pop(-mark-)tprint}readonly def/dicttype{pop
37 (-dictionary-)tprint}readonly def/nulltype{pop(-null-)tprint}readonly def
38 /filetype{pop(-filestream-)tprint}readonly def/savetype{pop(-savelevel-)
39 tprint}readonly def/fonttype{pop(-fontid-)tprint}readonly def/nametype{dup
40 xcheck not{(/)tprint}if cvsprint}readonly def/stringtype{dup rcheck{(\()tprint
41 tprint(\))tprint}{pop(-string-)tprint}ifelse}readonly def/arraytype{dup rcheck
42 {dup xcheck{((){tprint(typeprint}forall())tprint}{([]tprint(typeprint}forall())
43 tprint}ifelse}{pop(-array-)tprint}ifelse}readonly def/packedarraytype{dup
44 rcheck{dup xcheck{((){tprint(typeprint}forall())tprint}{([]tprint(typeprint}
45 forall())tprint}ifelse}{pop(-packedarray-)tprint}ifelse}readonly def/courier
46 /Courier findfont 10 scalefont def end errordict/handleerror{systemdict begin
47 $error begin $brkpage begin neverror{/newerror false store vmstatus pop pop 0
48 ne{grestoreall}if errorname(VMerror)ne{showpage}if initgraphics courier setfont
49 lmargin 720 moveto errorname(VMerror)eq{userdict/ehsave known{clear userdict
50 /ehsave get restore 2 vmreclaim}if vmstatus exch pop exch pop PrtVMMsg}{
51 (ERROR:)prnt errorname prnt nl(OFFENDING COMMAND:)prnt/command load prnt
52 $error/ostack known{nl nl(STACK:)prnt nl nl $error/ostack get aload length{==}
53 repeat}if}ifelse systemdict/showpage get exec(%%[Error:)print errorname
54 =print(; OffendingCommand:)print/command load =print( ]%%)= flush}if end end
55 end)dup 0 systemdict put dup 4 $brkpage put bind readonly put/currentpacking
56 where{pop/setpacking where{pop oldpack setpacking}if}if
57 %%EndResource
58 %%BeginResource: procset Pscript_WinNT_Full 5.0 0
59 /Pscript_WinNT_Full 300 dict 2 copy userdict 3 1 roll put dup begin
60 %%BeginResource: file Pscript_FatalError 5.0 0
Zeile 26, Spalte 78, C0 DOS Änd: 24.03.2012 11:29:30

Der gleiche Text, *medienneutral* aufbereitet:

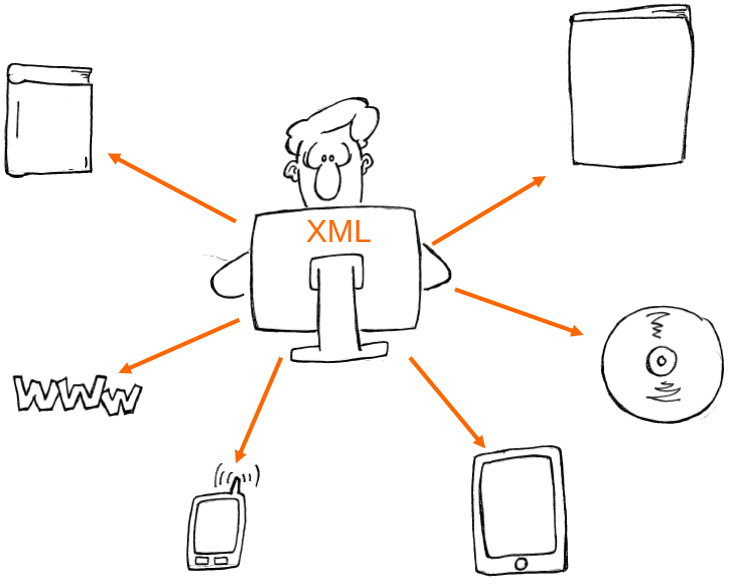
```
<absatz>
  Zutaten:<br/>
  1 frisches Huhn
</absatz>
```

Element { `<TAG>` … `</TAG>`

Der gleiche Text, medienneutral und semantisch aufbereitet:

```
<rezept typ="fleisch" kcal="480">
  <zutaten>
    <zut-zeile>
      <menge>1</menge>
      <zutat>frisches Huhn</zutat>
      <alt-zutat>TK-Huhn</alt-zutat>
    </zut-zeile>
  </zutaten>
</rezept>
```


Der moderne, contentzentrierte Workflow (viel besser)



Verschiedene Arten von Markup

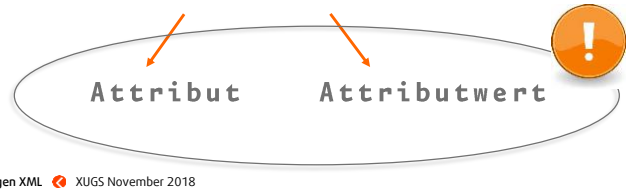
Der Löwe frisst die Gazelle.

Der $$Löwe@@ frisst die $$Gazelle@@.

```
Der <tier>Löwe</tier> frisst die <tier>Gazelle</tier>.
```

Der $$Löwe@@ frisst die %%Gazelle&&.

```
Der <tier typ="fleischfresser">Löwe</tier>
frisst die <tier typ="pflanzenfresser">Gazelle</tier>.
```



Die Auszeichnungstiefe in XML

Der Löwe frisst die Gazelle.

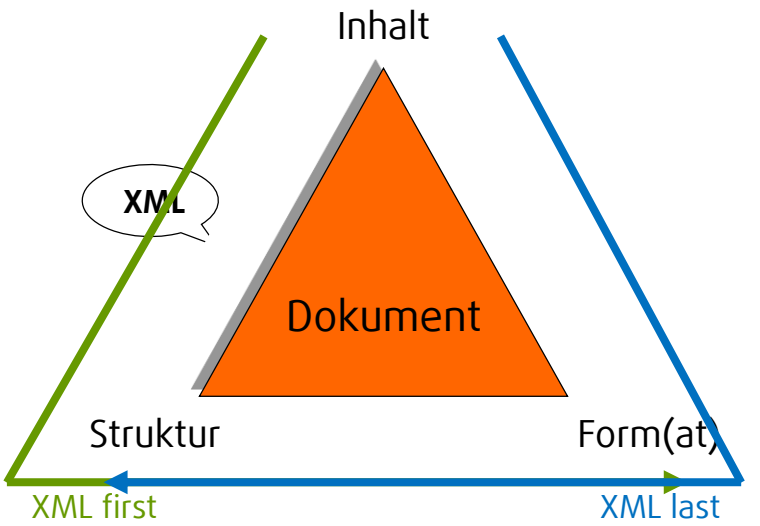
```
<absatz>
  Der Löwe frisst die Gazelle.
</absatz>


<absatz>
  Der <tier>Löwe</tier> frisst die <tier>Gazelle</tier>.
</absatz>


<absatz>
  Der <tier typ="fleischfresser">Löwe</tier>
  frisst die <tier typ="pflanzenfresser">Gazelle</tier>.
</absatz>
```


<prozedurales Markup>

```
<absatz size="10pt">
  Der <tier typ="fleischfresser" vorkommen="afrika"
  farbe="gelb" bedroht="ja" jäger="ja" lieblingstier="ja">
  Löwe</tier> frisst die
  <tier typ="pflanzenfresser" vorkommen="afrika"
  beutetier="ja" lieblingstier="nein">Gazelle</tier>.
</absatz>
```

<deskriptives Markup>

<hermeneutisches Markup>


Inhalt
XML
Dokument
Struktur
Form(at)
XML first
XML last

Getaggte Daten machen noch keine Struktur:
Die Baumstruktur von XML-Dokumenten
http://diepresse.com/images/uploads_425/2/d/9/320217/tar20070730131012.jpg

Die Syntax-Bestandteile von XML am Beispiel

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beispiel SYSTEM "verlagsstruktur.dtd">
<beispiel>
   <element attribut="attributwert">
      <!-- Hier steht ein Kommentar.
       Er wird in der Verarbeitung ignoriert -->
       Text, ggf. von weiteren <element2>Elementen</element2>
       unterbrochen.
   </element>
   <element>Auch XML enthält &quot;Entities&quot;.
       Dann gibt es noch sogennannte Processing Instructions.
       Mit ihrer Hilfe <?name steueranweisung?> kann man
       Befehle (z.B. für ein Satzsystem) in die Daten
       einbringen.
       Die Processing Instructions, kurz PI, werden ebenso
       wie die <!--Kommentare--> beim Parsen (=Prüfen gegen
       die Regeln einer Grammatik) ignoriert.
    </element>
</beispiel>
```

Mit Hilfe der Bausteine von XML –

***Elemente*, *Attribute*, *Entities*,**

***Processing Instructions* und *Kommentare* –**

können wir jedes Textdokument vollständig beschreiben.

Aber es bedarf noch weiterer Vorgaben, um aus einem „getaggten Text“ echtes XML zu machen:

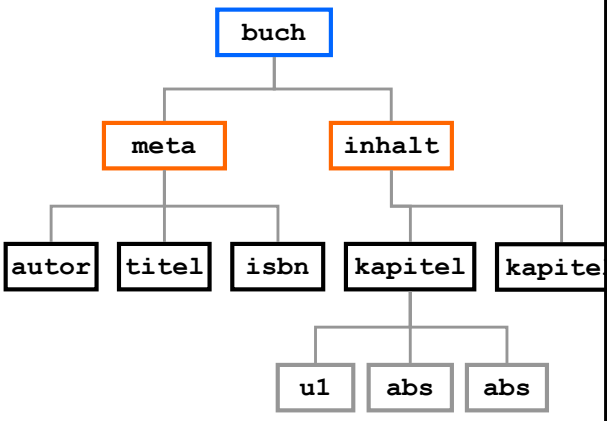
1. Die Bedingung der Wohlgeformtheit
2. Die „quasi-Bedingung“ der Validität
3. Die (empfohlene) Befolgung von Konventionen

Die wichtigsten Kriterien der „Wohlgeformtheit“

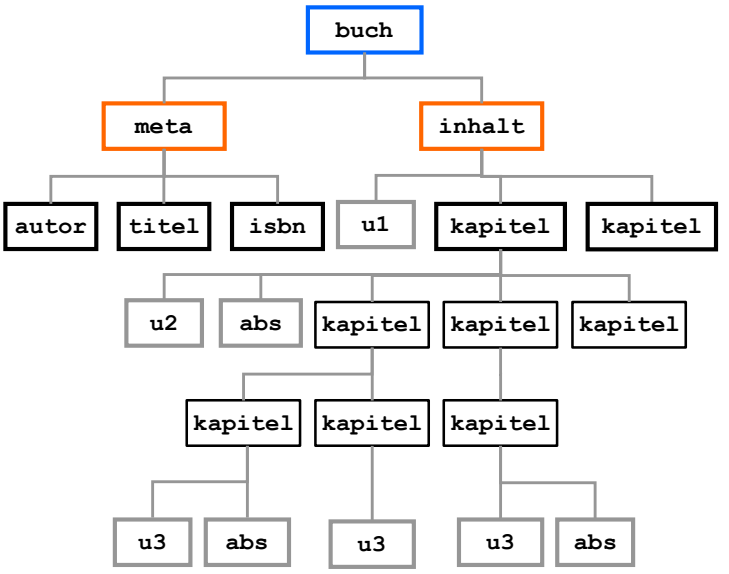
1. Es gibt genau ein Wurzelelement
2. Alle Elemente, die nicht leer sind, müssen mit Start- und Endetag gekennzeichnet sein. (`<abc>` … `</abc>`)
3. Leere Elemente enden mit einem Slash (<…/>)
4. Streng hierarchische Anordnung der Elemente erforderlich, keine Überlappungen erlaubt
5. Elementnamen sind "case-sensitive", d.h. die Groß/Kleinschreibung muss in Start- und Endetag gleich sein.
6. Attributwerte müssen in Anführungszeichen stehen
7. Ein Attribut darf nicht zweimal in ein und demselben Element auftreten
8. Alle verwendeten Entities müssen deklariert sein.

Die **Baumstruktur** in XML-Dokumenten

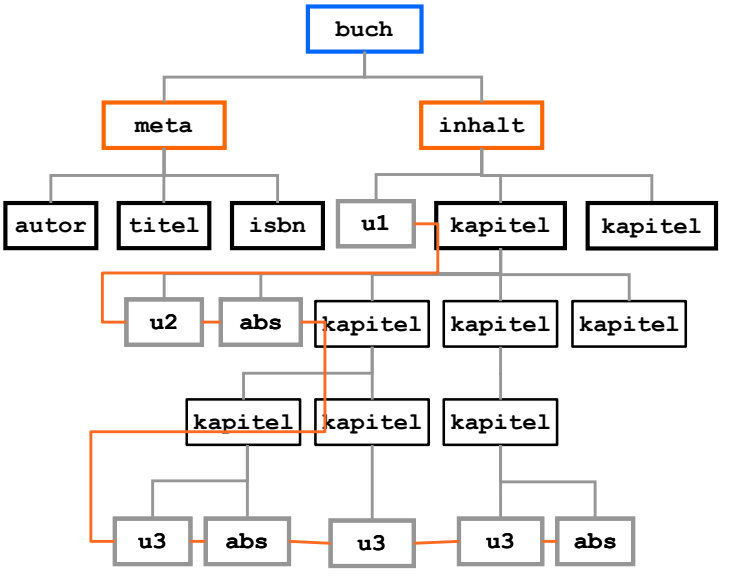
```
<buch>
  <meta>
    <autor> … </autor>
    <titel> … </titel>
    <isbn> … </isbn>
  </meta>
  <inhalt>
    <kapitel>
      <u1> … </u1>
      <abs> … </abs>
      <abs> … </abs>
    </kapitel>
    <kapitel>
      (…)
    </kapitel>
  </inhalt>
</buch>
```



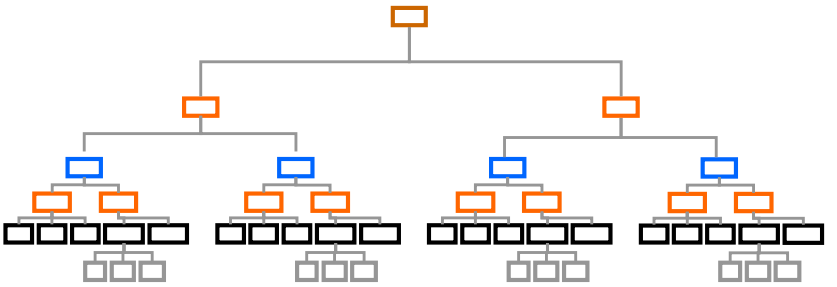
Komplexe Dokumente werden beherrschbar



Die Dokumentstruktur in InDesign / Word



Die Baumstruktur ist nicht zwingend auf ein Buch beschränkt:



Daraus leitet sich die nächste Forderung ab:

Die Inhalte müssen werkübergreifend(!) einheitlich strukturiert sein

...mit Hilfe einer **Grammatik**!

3. XML

kann leicht überprüft werden

Die Grammatiken in XML:
DTD & XSD

Abgrenzung

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!– DTD für Gedichte 20.03.2012 T. Ott –>
3
4  <!ELEMENT impressum EMPTY >
5
6  <!ELEMENT gedichtband  (impressum, vorwort, gedicht+, nachwo
7  <!ELEMENT vorwort     (#PCDATA)              >
8  <!ELEMENT gedicht     (autor, titel, strophe+)    >
9  <!ATTLIST gedicht     sprache     (de | en | el)   "de"
10                       e-jahr      CDATA        #REQUIRED
11                       schoen      (ja | nein)      #IMPLIED
12                       >
13
14 <!ELEMENT nachwort    (#PCDATA)              >
15 <!ELEMENT autor       (#PCDATA)              >
16 <!ELEMENT titel       (#PCDATA)              >
17 <!ELEMENT strophe     (vers+)                >
18 <!ELEMENT vers        (#PCDATA)              >
```

Was ist die DTD?

Die Dokumenttyp-Definition
= das Regelwerk (die “Grammatik”), nach dem meine XML-Datei aufgebaut ist

Eine Text-Datei (die Sie auch selbst mit einem beliebigen Texteditor erstellen können) mit einer festgelegten Syntax

XML-Grammatiken:
DTD *bzw.* XSD

Was ist eine XSD?

Die XML Schema Definition
= eine modernere und leistungsfähigere (aber auch komplexere) Form einer XML-Grammatik, in XML-Syntax verfasst

Die Rolle der Grammatik in XML

Die Regeln, nach denen ein XML-Dokument (»Instanz«) aufgebaut sein kann, werden in der zugehörigen Grammatik (Schema oder DTD) festgelegt.
Eine XML-Instanz ist mit einer Grammatik verbunden. Folgt die Instanz den Regeln dieser Grammatik, nennt man sie »valide«.

[XML-Instanzen ohne Schema/DTD, die jedoch den formalen Regeln von XML gehorchen, werden »wohlgeformt« genannt.]

In der DTD/dem Schema selbst werden definiert:

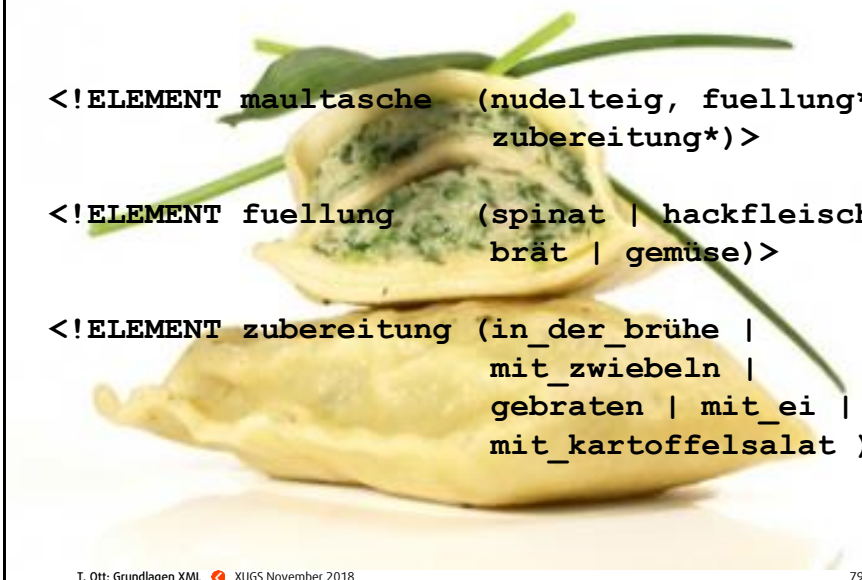
- Elementtypen, die im Text vorkommen können (wo und wie oft, verpflichtend oder optional, ...)
- ihre Attribute
- Entities, auf die hingewiesen werden kann

Die Struktur einer DTD: Das "Chicago Pizza Model"

```
<!ELEMENT pizza (base, tomatoSauce, topping*, cheese) >

<!ELEMENT base        EMPTY >
<!ATTLIST base type  (deepDish|thinCrust|stuffed) #REQUIRED >

<!ELEMENT topping    (pepperoni | mushrooms | ham | ...)+ >

<!ELEMENT tomatoSauce EMPTY >
<!ELEMENT cheese      EMPTY >
(...)
```

```
<pizza>
   <base type="thinCrust"/>
   <tomatoSauce/>
   <topping>
       <pepperoni/>
       <olives/>
       <tuna/>
   </topping>
   <cheese/>
</pizza>
```


Präzise Abbildung von Inhalten?

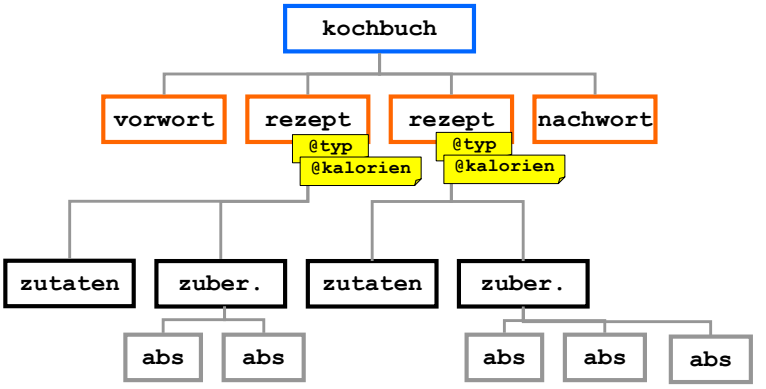


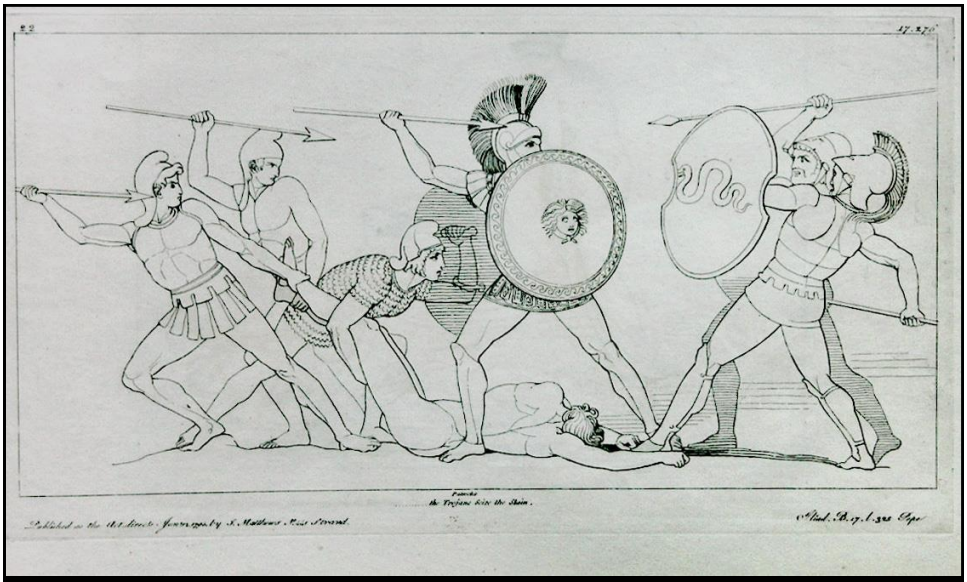
```
<!ELEMENT maultasche   (nudelteig, fuellung*,
                        zubereitung*)>

<!ELEMENT fuellung     (spinat | hackfleisch |
                        brät | gemüse)>

<!ELEMENT zubereitung (in_der_brühe |
                        mit_zwiebeln |
                        gebraten | mit_ei |
                        mit_kartoffelsalat )>
```

DTD für ein Kochbuch

```
<!ELEMENT kochbuch      (vorwort, rezept+, nachwort?)>
<!ELEMENT vorwort       (#PCDATA)                    >
<!ELEMENT rezept        (zutaten, zubereitung)       >
<!ATTLIST rezept        typ  (veg | fleisch) "fleisch"
                        kalorien CDATA  #REQUIRED    >
<!ELEMENT zutaten       (#PCDATA)                    >
<!ELEMENT zubereitung   (absatz+)                    >
<!ELEMENT absatz        (#PCDATA)                    >
<!ELEMENT nachwort      (#PCDATA)                    >
```

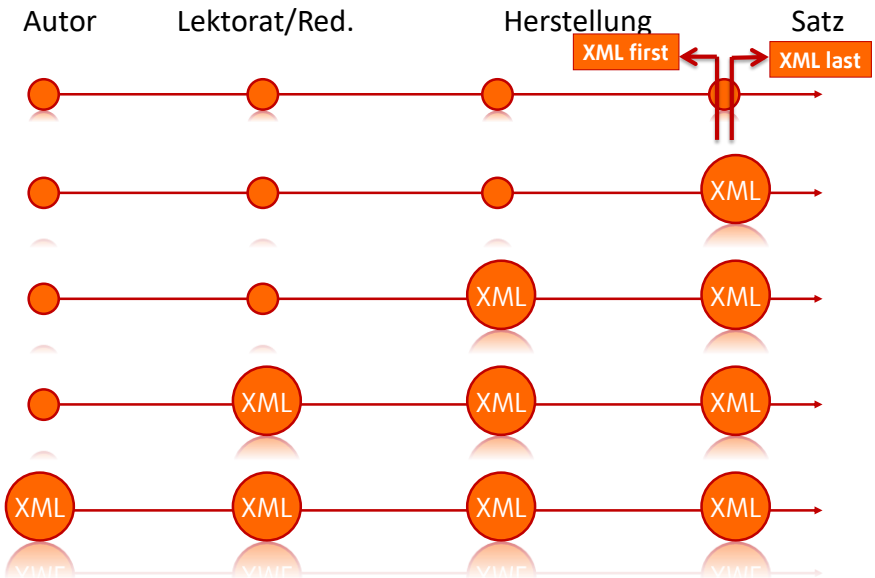

...Darstellung in der Baumstruktur



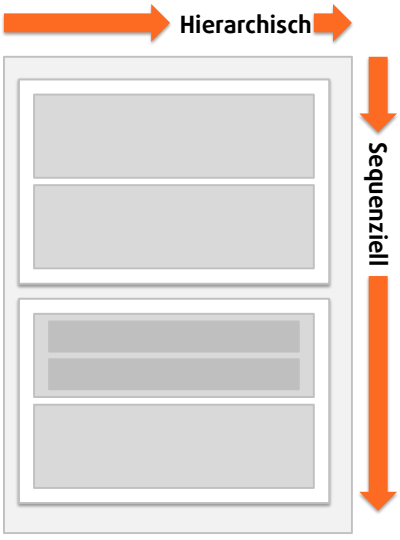


Konzepte im Wettstreit

XML first / XML last im Verlagsworkflow

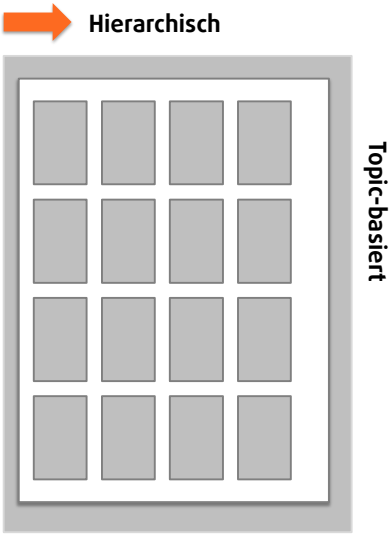


Das „Kapitel-Konzept“

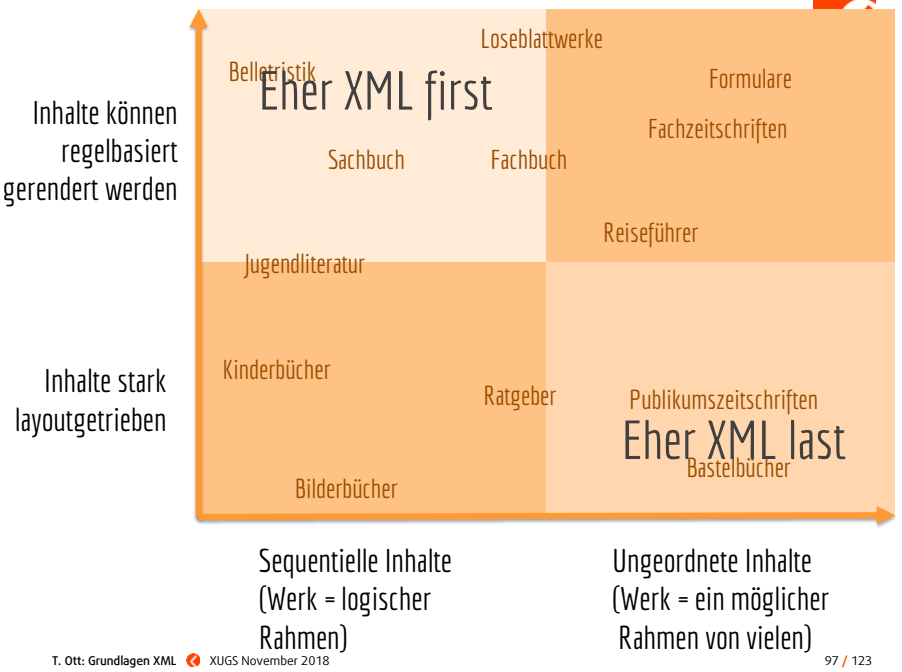


- Beschreibt die hierarchischen Abhängigkeiten
- Beschreibt die Lesereihenfolge
- Sequenz der Kapitel ist wichtig für das Verständnis des Ganzen, andere Zusammenstellungen sind nur schwer vorstellbar
- *Industriestandard: DocBook*

Das „Topic“-Konzept



- Geht von unabhängigen Informationseinheiten aus
- Definiert die „kleinste vermarktbare Einheit“
- Sequenz der Topics ist unerheblich, andere Zusammenstellungen sind gewünscht
- *Industriestandard: DITA*
  („Darwin Information Typing Architecture“)



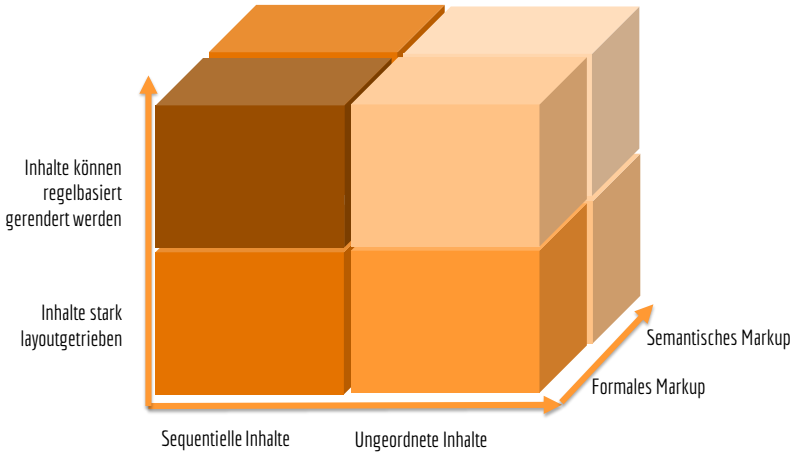
Der *strategische* Einsatz von XML in Verlagen bedeutet stets: SEMANTISCHES MARKUP

Also nicht:

```
<h1>Ravioli</h1>
<h2>Zutaten</h2>
<ul>
   <li>1 Dose Ravioli</li>
</ul>
<h2>Zubereitung</h2>
<p>warm machen</p>
```

Sondern:

```
<rezept typ="vorspeise">
   <titel>Ravioli</titel>
   <zutaten>
      <zutat>
         <menge>1</menge>
         <masseinheit>Dose</m
         <lebensmittel>Raviol
      </zutat>
   </zutaten>
   <zubereitung>
      <step>warm machen</step>
   </zubereitung>
</rezept>
```



Nochmals unser kleines

GEDANKENEXPERIMENT

zum Schluss:

Wenn man Verlage heute erfinden würde:

Würde sich die Frage nach XML first oder XML last überhaupt stellen?

Wäre die Frage nach der Produktionsmethode im Vordergrund?

Die gute Nachricht ist:

Es ist nicht einfach – aber einfacher, als Sie glauben

Es gibt mittlerweile viele „Best Practices“

Es gibt mittlerweile Standards

Es gibt günstige Softwarelösungen

Und:

Es gibt – endlich – den Diskurs über ein neues Selbstverständnis der Verlage im digitalen Zeitalter

Beteiligen Sie sich daran!