Software-Entwicklungswerkzeuge
Kap. 10 - Einführung

Prof. Dr. Uwe Aßmann
Technische Universität Dresden
Institut für Software- und Multimediatechnik
http://st.inf.tu-dresden.de
WS 11/12-0.3, 08.10.11

1) Taxonomie von Werkzeugen
2) Werkzeug-Grundtypen
3) Werkzeuglandschaft
4) Einführung in die Effektkategorien
5) Graph-Logik-Isomorphismus

Notwendigkeit des Einsatzes von Software-Entwicklungswerkzeugen

- **Kosten** der Softwareproduktion steigen ständig, weltweit > $ 250 Billionen im Jahr
- Wertschöpfung aus der **Softwareentwicklung** nach BMBF-Studie ca. 25,5 Mrd. EUR, bei Wachstumsrate von 12 % für 2003 etwa 38 Mrd. EUR
  - Bei Produkten der Telekommunikation und des Maschinen- und Anlagenbaus beträgt der Softwareanteil **75-80%** der **Herstellungskosten** *(steigend)*
  - Komplexe Vermittlungsanlagen bis zu **6000 Mannjahre**
  - Ein Mobiltelefon enthält ca. 250.000 lines of code (LOC)
- Mehr als 65% der **Berufstätigen** arbeiten mit dem Computer, 95% der verkauften Rechner ging in Haushalte, mehr als 400 Mio. Server im Internet.
- Die **Zuwachsraten im Softwaremarkt** liegen überdurchschnittlich hoch. Für
  - softwarebezogene Dienstleistungen 5,9%
  - Software 7,2%
  - davon Anwendungssoftware 8,8%
- **Wartungskosten** betragen etwa **60%** der Softwarekosten
- Aber: Nur ca. 30% der Unternehmen nutzen moderne Methoden und Werkzeuge.

Fehlerquellen bei der Software-Entwicklung

- Wichtig ist daher der Einsatz von Werkzeugen in frühen Phasen

Analyse:
Requirement falsch
Funktionale Spezif kation falsch

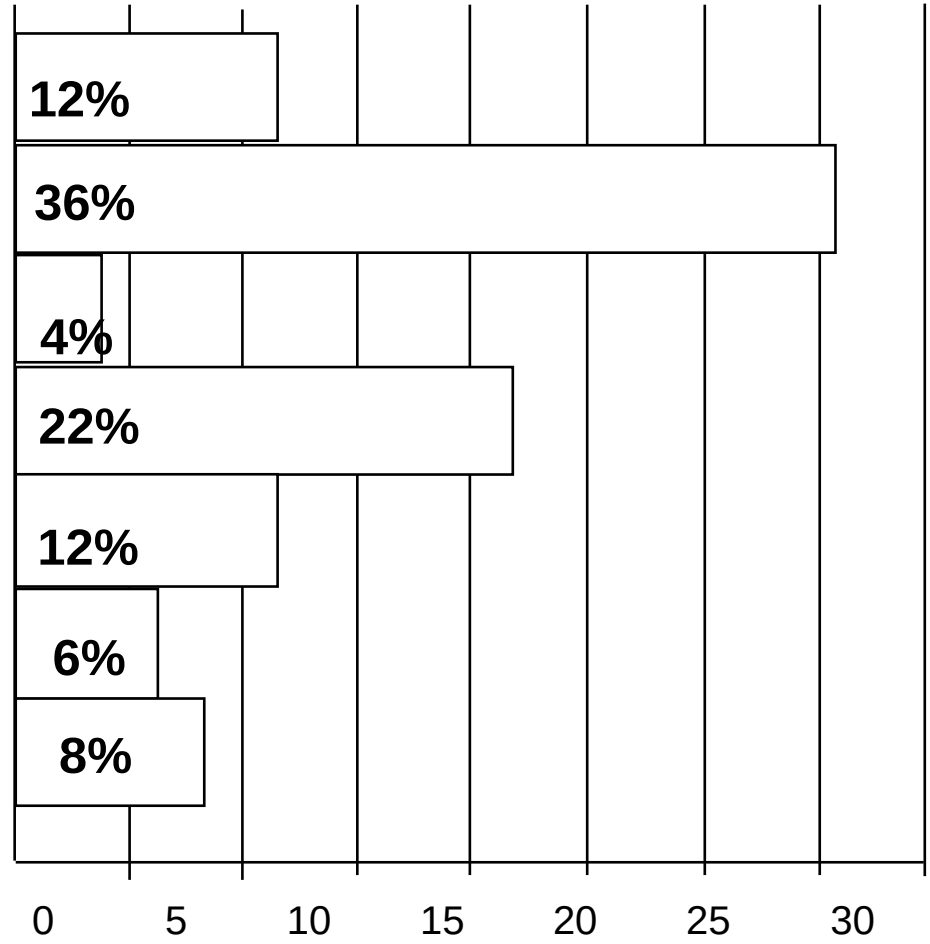
Entwurf:
Fehler in mehreren Komp.

Fehler in einer Komp.

Implementierung:
Denkfehler

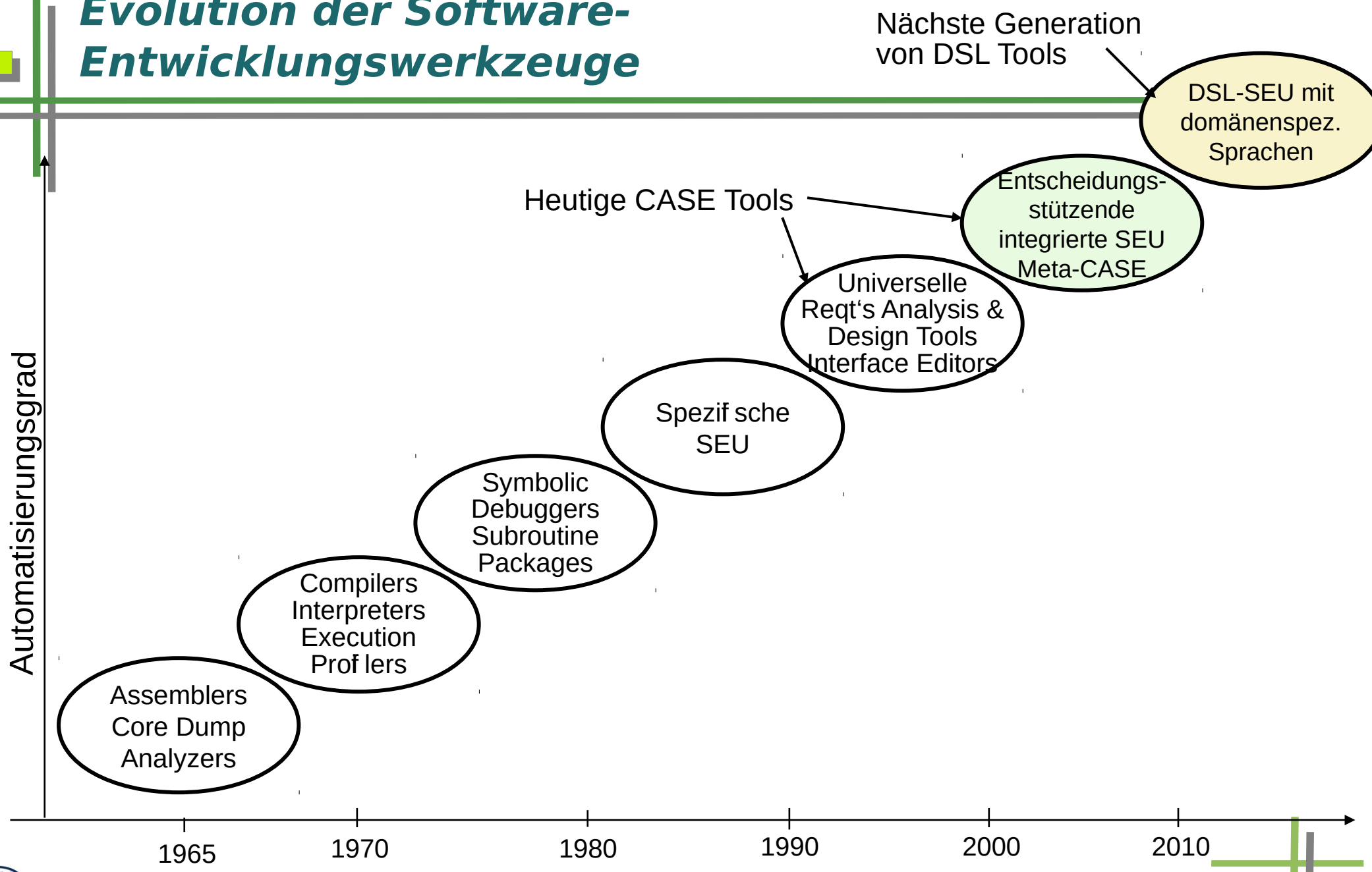
Fehler bei der Fehler-korrektur

Sonstige



Quelle: Balzert, H.(Hrsg.): CASE - Auswahl, Einführung, Erfahrungen; BI-Wissenschaftsverlag 1993, S.59

Evolution der Software-Entwicklungswerkzeuge



Quelle: nach [8 F91, S.20]

10.1 Taxonomie von Werkzeugen und Software-Entwicklungs-umgebungen (SEU)

10.0.1 Begriffs-Definitionen

Warum will der Mensch Werkzeuge einsetzen?

Ein **Werkzeug** ist ein Hilfsmittel, um Dinge schneller, präziser zu erledigen als von Hand.
Ein **IT-Werkzeug** ist ein Werkzeug, das im Rechner läuft und Informationen verarbeitet.
Ein **Software-Werkzeug** ist ein IT-Werkzeug, das Software bearbeitet.
Eine **Werkzeugmaschine** ist ein Werkzeug, mit dem man ein anderes Werkzeug herstellt.
Eine **Software-Werkzeugmaschine** ist ein Werkzeug, mit dem man andere Software-Werkzeuge herstellt.

- Werkzeuge werden eingesetzt
  - Zur Automatisierung
  - Zur Vereinfachung
- Extensive Werkzeugnutzung zeichnet den Menschen gg. allen anderen Lebewesen aus
- Werkzeuge können zum Bau von Werkzeugen eingesetzt werden
- Werkzeugmaschinen sind die Grundlage aller Produktivität
- Werkzeugmaschinen

“Tools and Material”-Metapher (TAM)

Tool:

- ist ein aktives Objekt, das Menschen benutzen können zum Umgestalten oder zum Verändern von **Material**, um eine spezifische Aufgaben zu lösen.
- **Tools** sind normalerweise geeignet für unterschiedliche Aufgabenbereiche, um verschiedenes Material zu bearbeiten.
- Viele konzeptuelle Eigenschaften der **Tools** können auf Software-Entwicklungswerkzeuge übertragen werden. Sie sollten für unterschiedliche Aufgaben und verschiedenes Material innerhalb von Softwaresystemen geeignet sein.

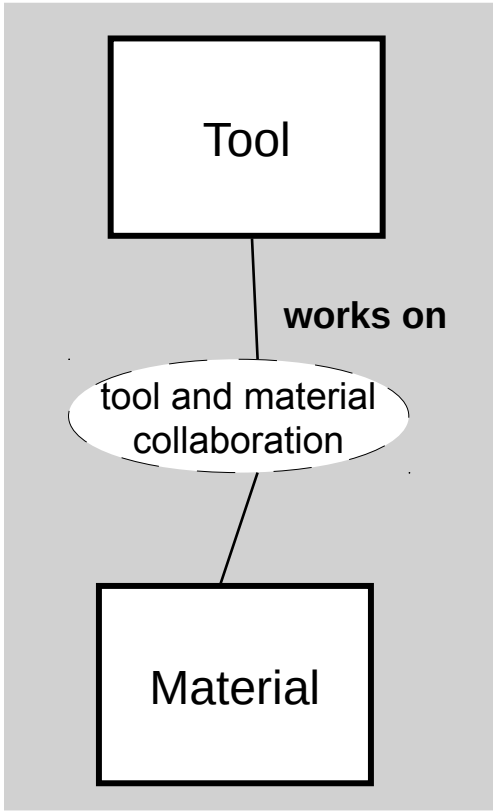
Material:

- ist ein passives Objekt, das Teil eines Arbeitsresultates wird. **Material** wird unter Benutzung von **Tools** verändert nach einem domänenspezifischen Konzept.
- Das Zusammenspiel von Tools und Material wird durch eine Kollaboration (Rollenmodell) ausgedrückt (siehe „Softwaretechnologie“).

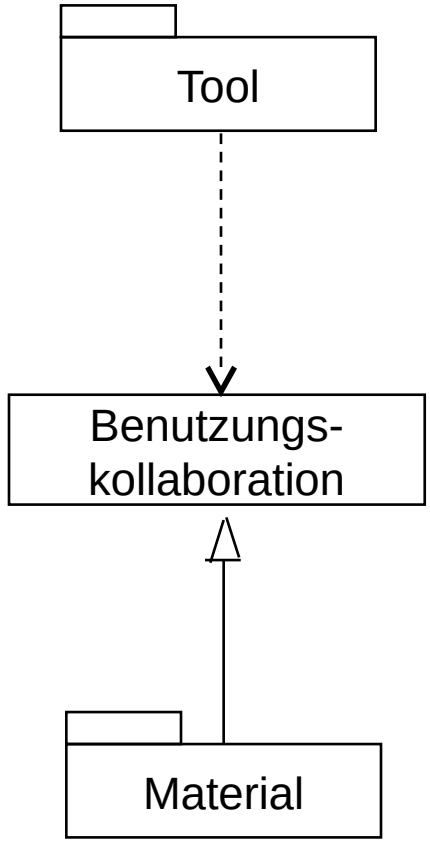
[Züllighoven, Heinz: Object-Oriented Construction Handbook; dpunkt.verlag 2005]

Diese Definitionen sind Basis einer allgemeinen Pattern Language for werkzeuggestützte Softwaretechnologie.
Siehe auch Kurs „Design Patterns and Frameworks“

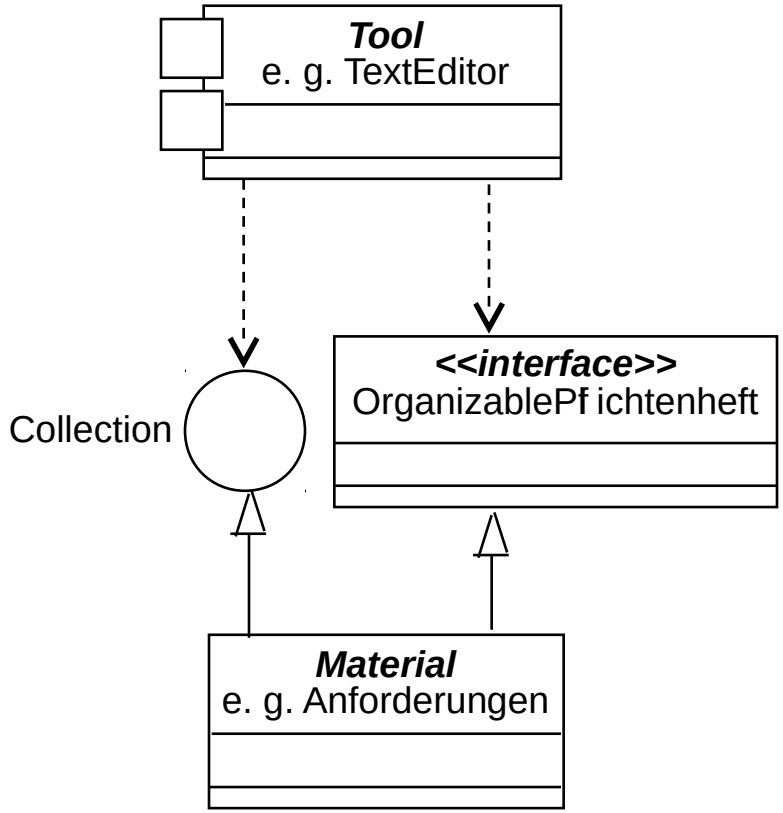
Tool and Material - Kollaboration



Conceptual Pattern



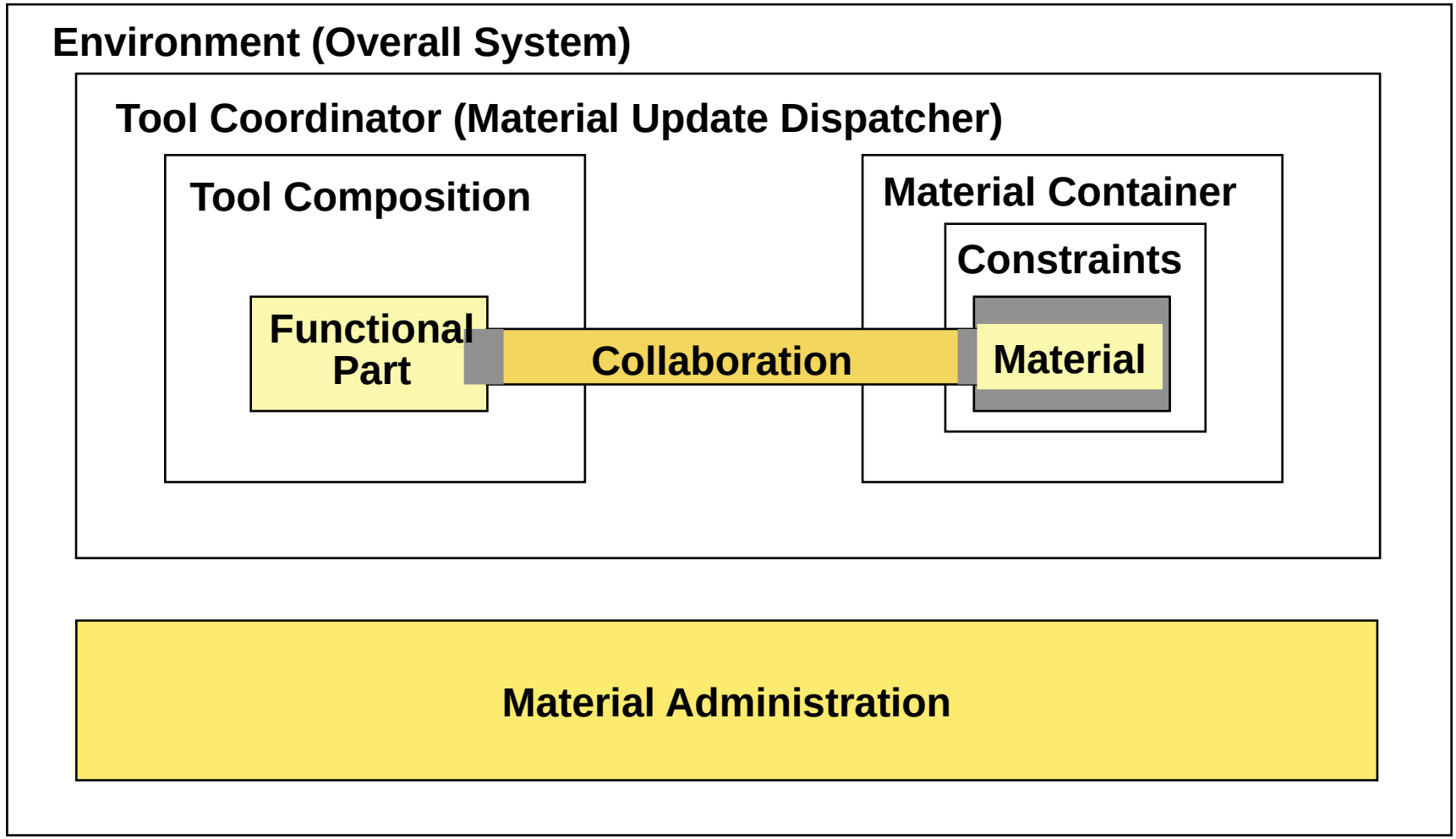
Design Pattern



Construction part

Quelle: Züllighoven, H.: Object-Oriented Construction Handbook; dpunkt.verlag Heidelberg 2005, S. 87

TAM Patterns for Tool Integration



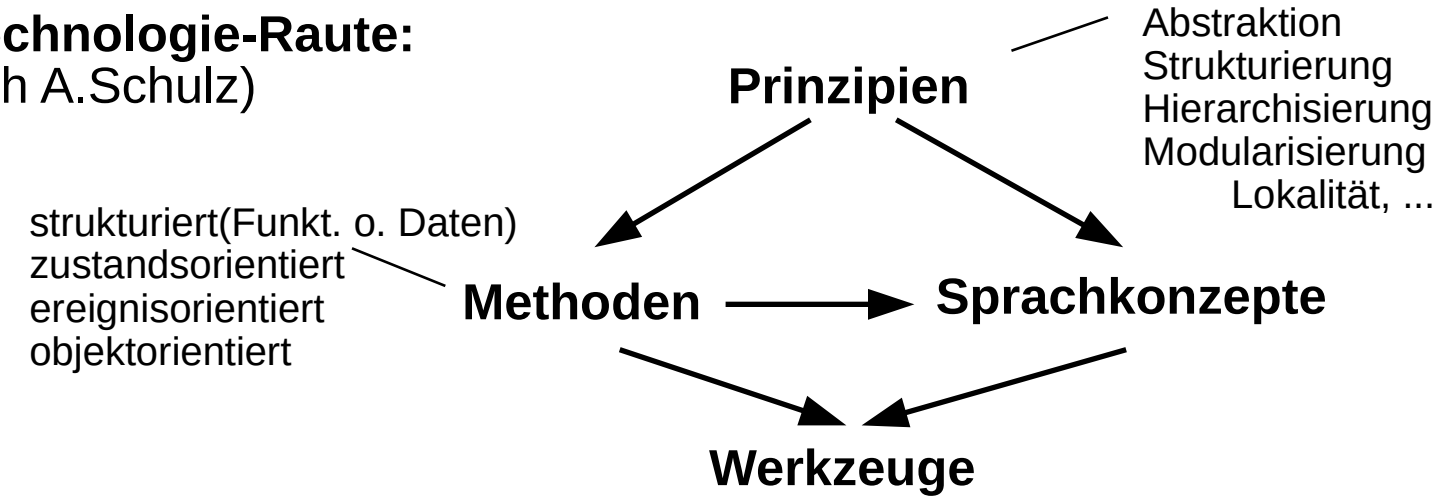
Quelle: Riehle, D., Züllighoven, H.: Pattern Languages of Program Design; Reading, Massachusetts: Addison Wesley 1995, Chapter 2, S. 9-42

Siehe auch Kapitel „Repository“

Definitionen

Software-Werkzeuge sind Programme (Software, Hilfsmittel), die Vorgehensweisen, *Prinzipien, Methoden* und *Sprachkonzepte* rechnergestützt umsetzen und den Benutzer bei der Software-Entwicklung unterstützen (nach [6, S.204])**.**

Softwaretechnologie-Raute:
(nach A.Schulz)



Software-Entwicklungsumgebungen (SEU)

Eine **Software-Entwicklungsumgebung (SEU)** besteht aus einer **strukturierten Menge integrierter Werkzeuge und Bausteine,** die ein Team bei allen in der Software-Entwicklung anfallenden Tätigkeiten unterstützen soll einschließlich einer einheitlichen Methodik für seine Nutzung.

- Eine SEU ist also eine komplexe Software-Werkzeugmaschine
  - Computer aided Software Engineering (CASE), CASE-Umgebung
    - CASE Environment
    - Integrated Computer Aided Software Engineering (I-CASE)
  - Software-Produktionsumgebung (SPU)
  - Software Engineering Environment System (SEES)
  - Integrated Project Support Environment (IPSE)
  - Integrated Software Engineering Environment (ISEE)
  - Integrated Software Factory (ISF)

Nagl. M.: Software-Entwicklungsumgebungen: Einordnung und zukünftige Entwicklungslinien; Informatik-Spektrum 16(1993) H.5, S. 273-280

Spannbreite des Begriffes SEU

umfasst:

(1) eine auf einen Anwendungsbereich abgestimmte **Modellierungs- oder Arbeitsumgebung**, in der der Anwender direkt seine Gedankenwelt vorf ndet und nicht mehr im klassischen Sinne programmiert;

(2) eine **Auswahl von Werkzeugen** und Bausteinen für einen Anwendungsbereich, die dort angewandte Methoden und Programmiersprachen unterstützen;

(3) eine **Sammlung** vorgegebener, mehr oder minder brauchbarer **Programmbausteine** für einen Anwendungsbereich;

(4) eine abgestimmte Softwaretechnik-Arbeitsumgebung zur **Erstellung beliebiger Softwaresysteme** auf eine oder mehrere Programmiersprachen abgestimmt;

(5) eine (CASE-)**Plattform** (Prozesskoordination, Objektspeicher, Kommunikationsmechanismen) **für SEU**, die auch für andere verteilte Anwendungen genutzt werden kann;

(6) eine **Meta-Umgebung** zum Bau von SEU **(Meta-CASE)**.

Quelle: Nagl. M.: Software-Entwicklungsumgebungen: Einordnung und zukünftige Entwicklungslinien; Informatik-Spektrum 16(1993) H.5, S. 273-280

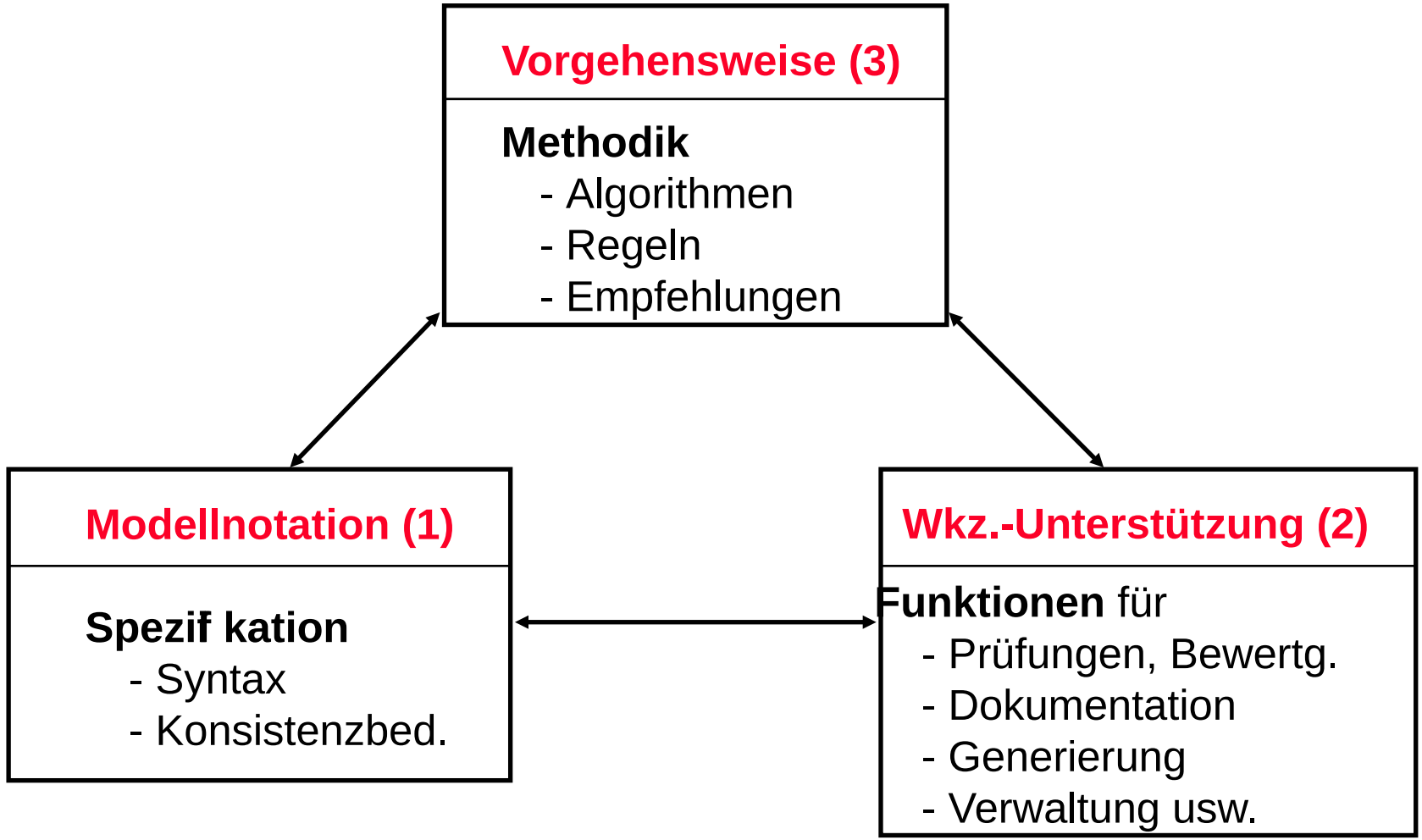
10.1.2 Aufbau und prinzipielle Funktion von Software-Entwicklungswerkzeugen

Werkzeugfunktionen

- Ursprünglich wurden nur einzelne grundlegende Komponenten der Software-Entwicklung wie Compiler, Editoren oder Testhilfen als Werkzeuge bezeichnet
- Im Laufe der Zeit kamen viele **spezialisierte** Entwicklungs- und Administrationswerkzeuge hinzu:
  - Herstellung und Verarbeitung von **Artefakten** oder **Dokumenten** (Prosa, Bilder, Diagramme, Modelle, Spezif kationen, formatierte Texte, Programme, Code)
    - **--> Modellnotation (1)**
  - Konsistenzprüfung auf Wohlgeformtheit von einzelnen Dokumenten und zusammengehörigen Dokumentenbeständen, Produktverwaltung während der Herstellung und Wartung
    - **--> Werkzeugprüfung/-Unterstützung (2)**
  - Unterstützung von Methoden und einzelner Entwicklungsschritte (Entwurf, Testen,...)
  - Unterstützung von Phasen- und Vorgehensmodellen
    - **--> Vorgehensweise/Methodik (3)**

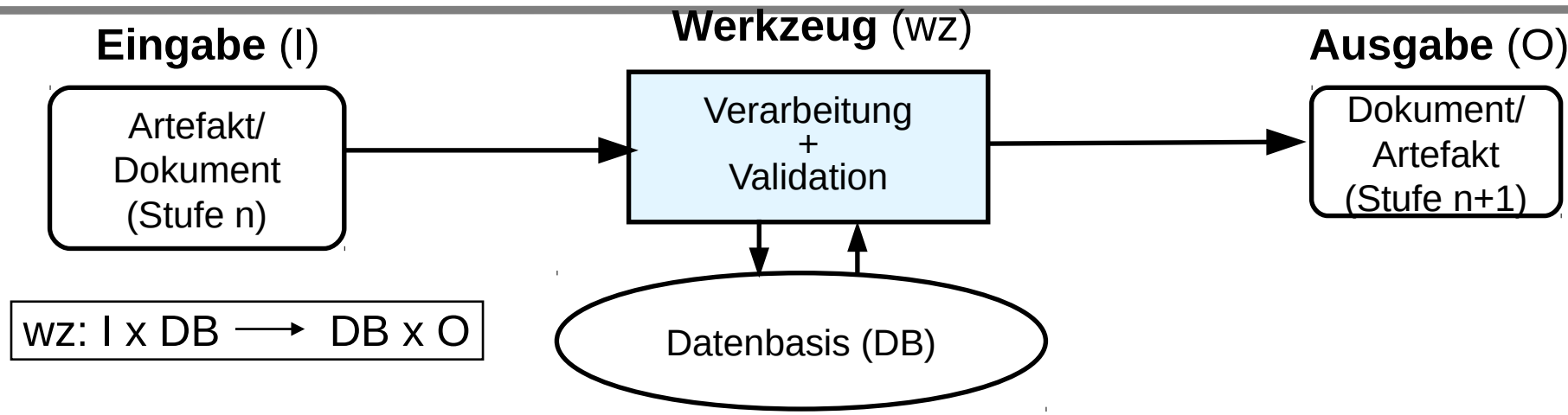
In Werkzeugen unterstützte Aspekte

(Auch Modellaspekte der Basistechniken)

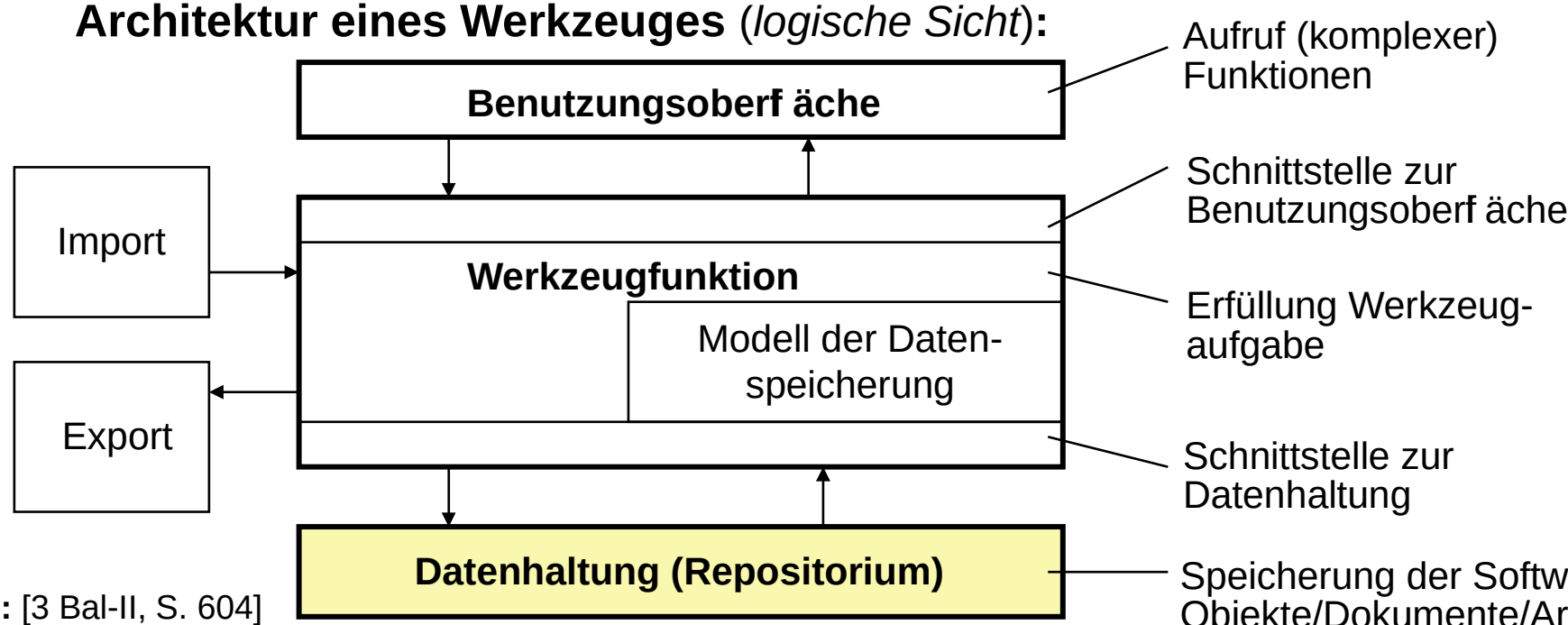


Quelle: nach Raasch, J.: Systementwicklung mit strukturierten Methoden; Hanser Verlag (2. Auf age) München 1992

Werkzeug - Wirkungsschema

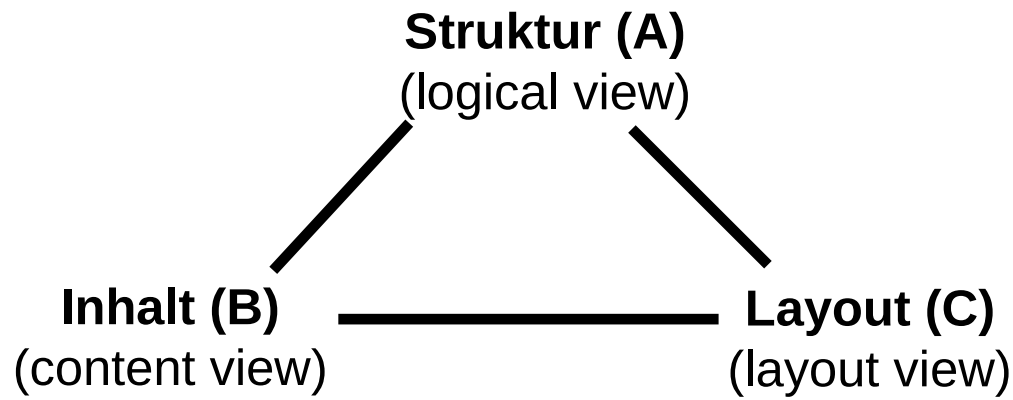


Architektur eines Werkzeuges (*logische Sicht*)**:**



Quelle: [3 Bal-II, S. 604]

Aspekte von Dokumenten/Artefakten



Struktur: log. Einheiten, wie Gliederung, Überschriften, Fußnoten, Köpfe, Verweise
kontextfreie Struktur
kontextsensistive Struktur (statische Semantik)
Inhalt: (Kapitel-)Text, Graf ken, Bilder, Bitmuster, elektron. Erscheinungsformen
Layout: Ausgabeanordnungen und -vorschriften für log. und inhaltliche Elemente

---

Standards, die Aspekte von Dokumenten trennen:
SGML = Standard Generalized Markup Language (Teilmenge ist HTML).
Legt in abstrakter Form die Struktur von Dokumenten fest.
XML = Extensible Markup Language - Stukturbeschr. mit DTD--> XML-Tags

Dokumenttypen (Artefakte) der Softwareentwicklung

- Text
  - z.B. Anforderungsspezifikation, Entwurfsspezifikation, Programmbeschreibungen,...
- Grafik
  - z.B. Analyse- und Entwurfsspezifikation (UML-Diagramme), Programmstrukturen,...
  - komplexe visuelle Darstellungen in 2-D oder 3-D
- Code
  - z. B. Pseudocode, Codegerüste, Quellcode
- Tabellen
  - z.B. Relationen, Testfalltabellen

Eigenschaften von Softwareentwicklungs-Dokumenten:

Struktur	• **Struktur** meist vorgegeben (UML), Standardisierungsgrad wächst
	• haben einen **Autor** (oder mehrere bei Gruppenarbeit),
	• und verschiedene **Zielgruppen**,
Inhalt	• sind **komplex** und **umfangreich**,
	• müssen **präzise** sein (genaues Abbild des Originals, Formalisierung),
	• durchlaufen einen **Entwicklungszyklus**,
Layout	• sollen **lesefreundlich** und "**schön**" aussehen (Verständlichkeit),
	• sind Gegenstand von **Reviews**,
	• werden **maschinell** erzeugt und geprüft (Validierung).

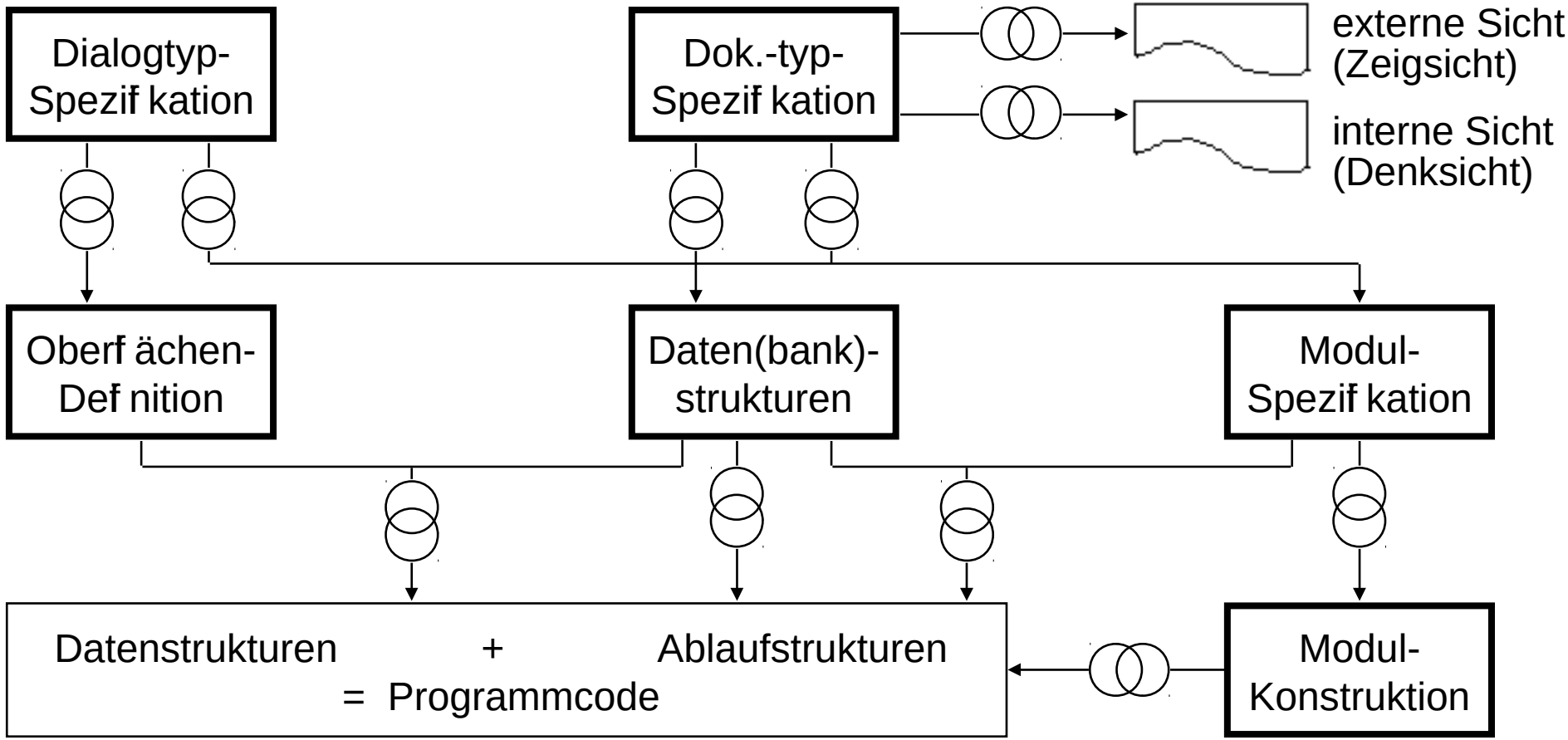
Werkzeuggetriebene Softwarentwicklung

Dokumentenorientiert (artefaktor.)
Dokument als primäres Endprodukt steht ständig im Vordergrund
Entwickler schreibt Software
typisch ist eindimensionaler Text, Graf ken werden eingeführt
Hauptwerkzeug ist universeller (Text-) Editor
Methodik (Metamodell) f exibel
ganzheitliche Denkweise ausgerichtet am Dokument
klare Abgrenzung der Verantwortlich-keit für Dokumente (evtl. Gruppenab-stimmung)

transaktionsorientiert
interaktive Entwicklung von Artefakten steht im Vordergrundi
Entwickler zeichnet Software
typisch sind zweidimensionale Graf ken, Text eingefügt
mehrere methodenorientierte graf sche Editoren
Methodik (Metamodell) durch CASE starr vorgegeben
atomares Agieren der Elemente in der Datenbank
Verantwortungsabgrenzung für graf sche Elemente schwieriger

Quelle : Denert, E.: Dokumentenorientierte Software-Entwicklung; Informatik-Spektrum 16(1993) H. 3, S. 159 - 164

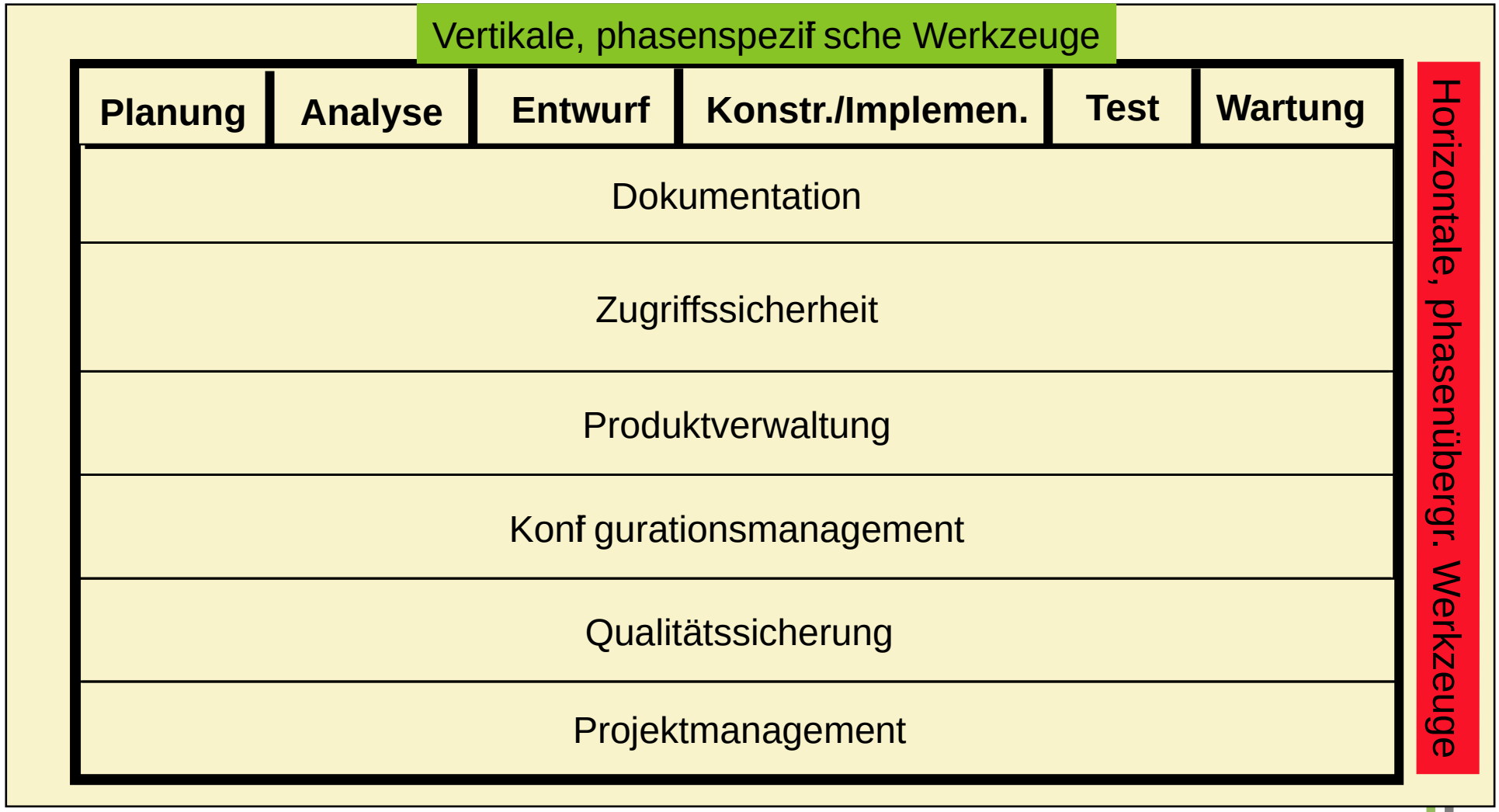
Dokumententransformation durch Werkzeuge



Quelle: Denert, E.: Dokumentenorientierte Software-Entwicklung; Informatik-Spektrum 16(1993) H. 3, S. 159 - 164

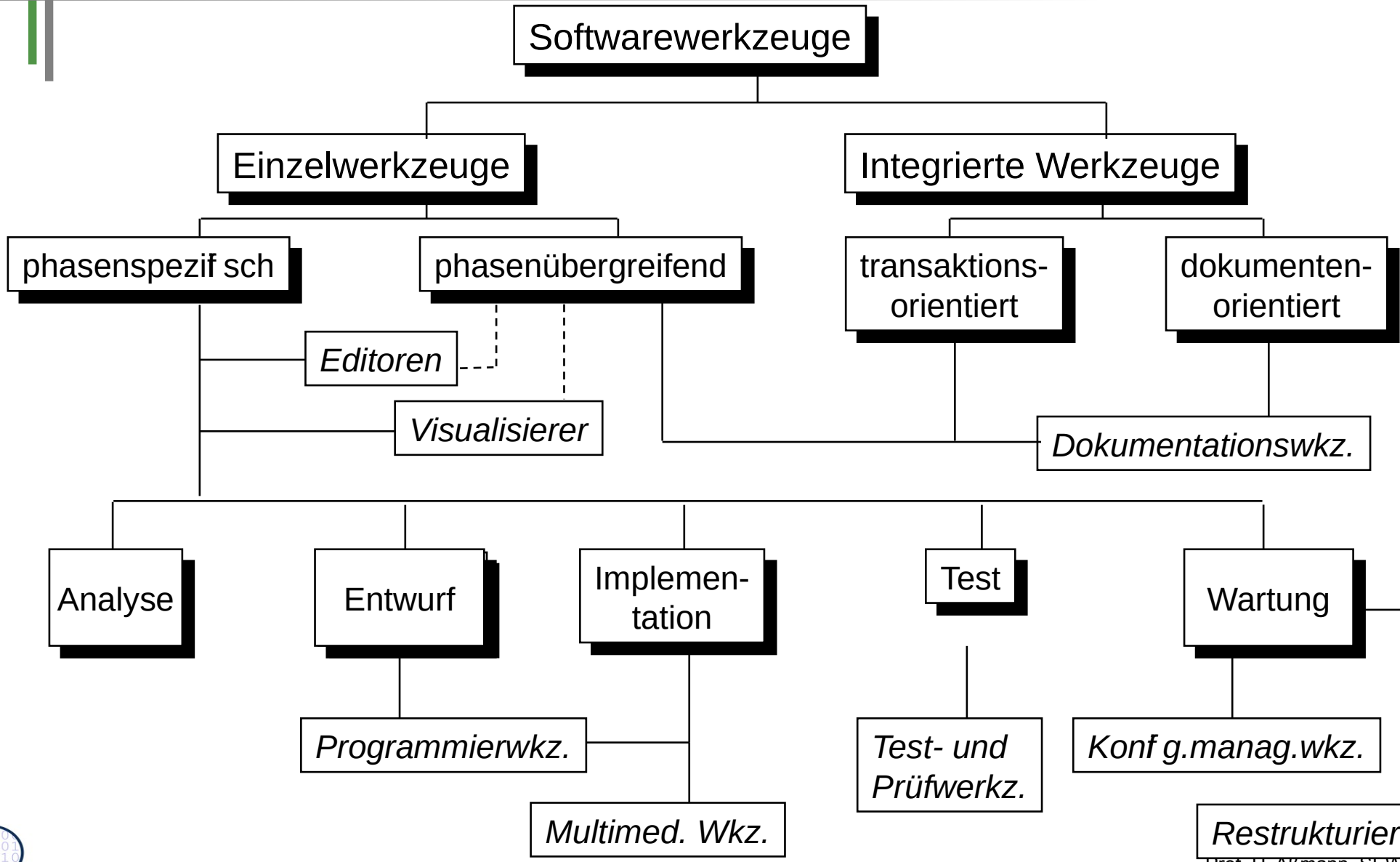
10.2 Werkzeuggrundtypen - Klassen von CASE-Tools

Entwicklungsaufgaben und Werkzeuge



Quelle: Forte, G., McCulley, K.(Hrsg.): CASE Outlook: Guide to Products and Services; CASE Consulting Group, Lake Oswego, Orl., 1991

Eine Grobgliederung von Software-Entwicklungswerkzeugen



Bespiele für phasenspezifische Wz: Editoren

Aufgabe:	Interaktive Bearbeitung (Erstellen, Modif zieren, Inspizieren) von Spezif kationen(Code, Text, Graf k, Bild usw.)
Modell:	die zu bearbeitenden Spezif kationen und Medienobjekte (Daten), die neutral (universelle Editoren) oder spezif sch (sprachorientiert, syntaxunterstützend) sein können.
Sichten:	Präsentation in der Regel statisch z. B. in Bildschirm-fenstern, die alle Daten anzeigen
Interaktion:	Benutzereingaben (Tastendrücke, Mausklicks o.ä.) reagie-ren auf Eingabeteile(Texte, Programme, visuelle Darstel-lungen)
Struktur des Modells:	je nach Objektdef nition, Objektassoziationen, Methoden-struktur, Struktur des Vorgehens
Zeitpunkt:	in der Regel asynchron, synchron bei Teamarbeit
Initiative:	benutzerorientiert
Arten:	Texteditoren, Spracheditoren, Figureneditoren, Pixeleditoren

Bespiele für phasenübergreifende Wz:
Browser

Aufgabe:	Visualisierung externer und interner Strukturen zum selektiven Lesen großer Datenmengen, die in übersichtlicher und strukturierter Form aufbereitet werden (to browse = schmökern) auch mit Include-Funktionen
Modell:	der übersichtlich darzubietende Informationsraum z. B. in Form von Hierarchien, Objekten, Klassen, Methoden bzw. Dokumenten
Sichten:	Präsentation in der Regel statisch auch dynamisch in Bildschirmfenstern, die verständlichen Modellausschnitt anzeigen
Interaktion:	Navigieren und Explorieren in großen Strukturen auch unter Einschaltung von Auswahl- und Filterfunktionen
Struktur des Modells:	externe Objekt-/Methodenstruktur aber auch dynamische Methodenstrukturen
Zeitpunkt:	Visualisierung synchron während Programmlauf, seltener asynchron nach dem Programmlauf
Arten:	Datei-Browser, Symbol-Browser, Hierarchie-Browser, Netz-Browser, Schnittstellen-Browser

Bespiele für phasenübergreifende Wz:
Dokumentationswerkzeuge

Aufgabe:	Variable Aufbereitung von Auswertungsinformationen zur Erzeugung, Vervielfältigung, Ablage und Sammlung von Softwaredokumentationen (online-Handbüchern)
Modell:	Aufbau der Softwaredokumente nach Inhalt, Struktur, Layout, die variabel(Beschreibungssprachen) aufbereitet und ausgegeben werden müssen
Sichten:	Bereitstellung unterschiedlicher Sichten und Formate im Bildschirmfenster und WYSIWYG auf Drucker, Plotter
Interaktion:	Interaktive variable Aufbereitung im Dialog, Dokumenten-produktion im Batchbetrieb
Struktur des Modells:	je nach gewählten Firmenvorschriften, Standards, z. B. DOD-STD-2167A (Defense System Software Development)
Zeitpunkt:	in der Regel asynchron zeitversetzt
Initiative:	benutzerorientiert in der Aufbereitung und bei Abarbeitung systemgesteuert
Arten:	Textverabeitung(Word), Desktop Publishing(FrameMaker), Dokumentengeneratoren

Bespiele für phasenspezifische Wz:
Generatoren

Aufgabe:	Transformation von graf schen und/oder textuellen (Entwurfs-)Spezif kationen in eine höhere Formalisierung wie Code, Codegrüste, Programmteile oder ganze Programme
Modell:	Syntax und Semantik der Eingabesprache, aus der über feste Abbildungsregeln in die gewünschten Ausgabespezif kationen zu erzeugen sind
Sichten:	statische Eingabeströme, die über Steuerparameter angepasst werden können
Interaktion:	Visualisierung von Fehlersituationen mit der Möglichkeit des direkten Eingreifens im Dialogbetrieb
Struktur des Modells:	je nach Def nition der Eingabesprache und der daraus zu generierenden Ausgabe-Objekte
Zeitpunkt:	in der Regel asynchron, Ausgabe versetzt nach Korrektur
Initiative:	Systeminitiative, nur benutzerorientierte Eingabeaufbereitg.
Arten:	Programmgeneratoren, Masken-Generatoren, Test(fall-) Generatoren, Report-Generatoren

Bespiele für phasenspezische Wz: Debugger

Aufgabe:	Steuerung, Anzeige und Protokollierung der Zustände zu testender Programme in lesbarer Form
Modell:	die Struktur des Testprogramms mit def nierten Mengen fester sowie variabler Haltepunkte
Sichten:	Anzeige der momentanen Stelle im Programm, des Inhalts lokaler und globaler Variabler und in Ausführung bef nd-licher Prozeduren durch Transformation des Maschinen-zustandes in den zugeh. Zustand des Quellprogramms
Interaktion:	Benutzereingaben zur Reaktion auf online Debugger-Aus-gaben, Manipulation in Programm-(Objekt-)Strukturen
Struktur des Modells:	gebildet durch Referenzinformationen(Namen, Typen Adressen von Variablen und Prozeduren)
Zeitpunkt:	synchron bei interaktiver Abarbeitung(*Tracing Debugger*) und asynchron bei *Post-Mortem* Aufruf
Initiative:	benutzerinitiiert bei *Tracing* und systeminitiiert bei *Post-M.*
Arten:	absolute Debugger(Masch.-Niveau), symbolische Debug-ger(Source-Code), graf sche Debugger

Klassen von CASE-Tools (1)

Klasse	Subklassen	Grundfunktion
Editor	Text-Editor Graphischer Editor	Erstellen und Modifizieren aller Arten von Spezifikationen und Medienobjekten
Visualisierer	Browser Viewer Inspektor	Visualisierung externer Strukturen und multimedialer Objekte(Hierarchien, Aufrufe bzw. Dokumente). Interaktion mit anderen Werkzeugen.
Programmierwerkzeuge	(Cross-)Assembler (Cross-)Compiler Interpreter Linker Generator Restrukturierer	Transformation textueller (und visueller) Programmspezifikationen in einen Zwischencode oder Maschinencode, der unmittelbar interpretiert oder aus dem Objektcode ausgeführt wird. Analyse der Syntax (und der Semantik). Erzeugung v. Codegerüsten aus sem. Spezifikat. u. umgekehrt.
Test- und Prüfwerkzeuge	Manipulatoren Debugger Tracer Stat. Analyzers Dyn. Analyzers Comperator	Syntaktische Überprüfung von Spezifikationen u. Programmen. Manipulation der Eingangsbedingungen und Festlegung des Testablaufes. Vergleich mit Sollbedingungen. Auswertung und Visualisierung der Ergebnisse nach unterschiedlichen Kriterien, Metriken

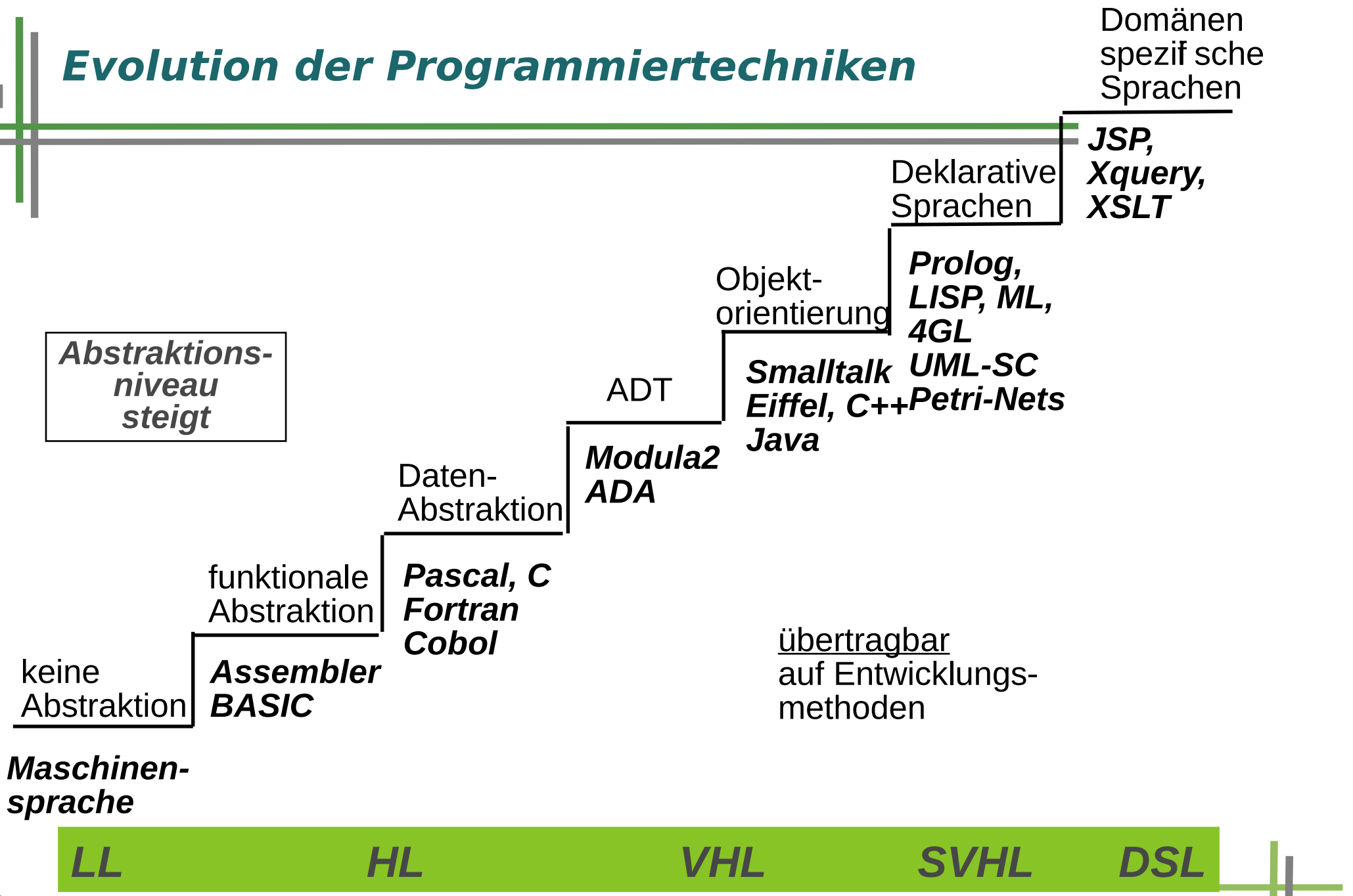
Quelle: Fugetta, A.: A Classifkation of CASE Technology; Computer 26(1993) H. 12, S. 25-38

Klassen von CASE-Tools (2)

Klasse	Subklassen	Grundfunktion
Test- und Prüfwerkzeuge	Simulator Testfall-Generator Testverwaltgs.werkz.	Interpretation der Ausführung von Spezif kation. Erzeugen von Testdaten aus der Progr.spezif k. Verwaltung aller Testresultate, Testreports
Konf g.manag.-Werkzeuge	Repositories Versionsverwalter Konf gurat.builder Produktverwalter	Ablage und Verwaltung sowohl anwendungs-bezogener als auch multimedialer Objekte. Über-wachung der Versions- und Konf gurationsbil-dung. Navigations- und Wiederauff ndungsfunkt.
Dokumentat.-Werkzeuge	Textverarbeitung Transformatoren Desktop Publishing Hypertextsysteme Spreadsheets	Variable Aufbereitung von Auswertungsinforma-tionen nach Struktur, Inhalt und Layout. Erzeu-gung von (online-)Handbüchern, Projektdoku-mentationen und Tutorials in unterschiedlichen Formaten.
Multimediale Werkzeuge	User Interface Builder Autorensysteme Synchronisatoren	Bildung und Integration dynamischer und multi-medialer Objekte in einer Applikation bestehend aus Präsentation und Interaktionssteuerung.
Wartungswkz.	Änderungskontrolle Fehlerbehebungshilf.	Überwachung aller Programmänderungen, Koordinierung der Zugriffsberechtigungen

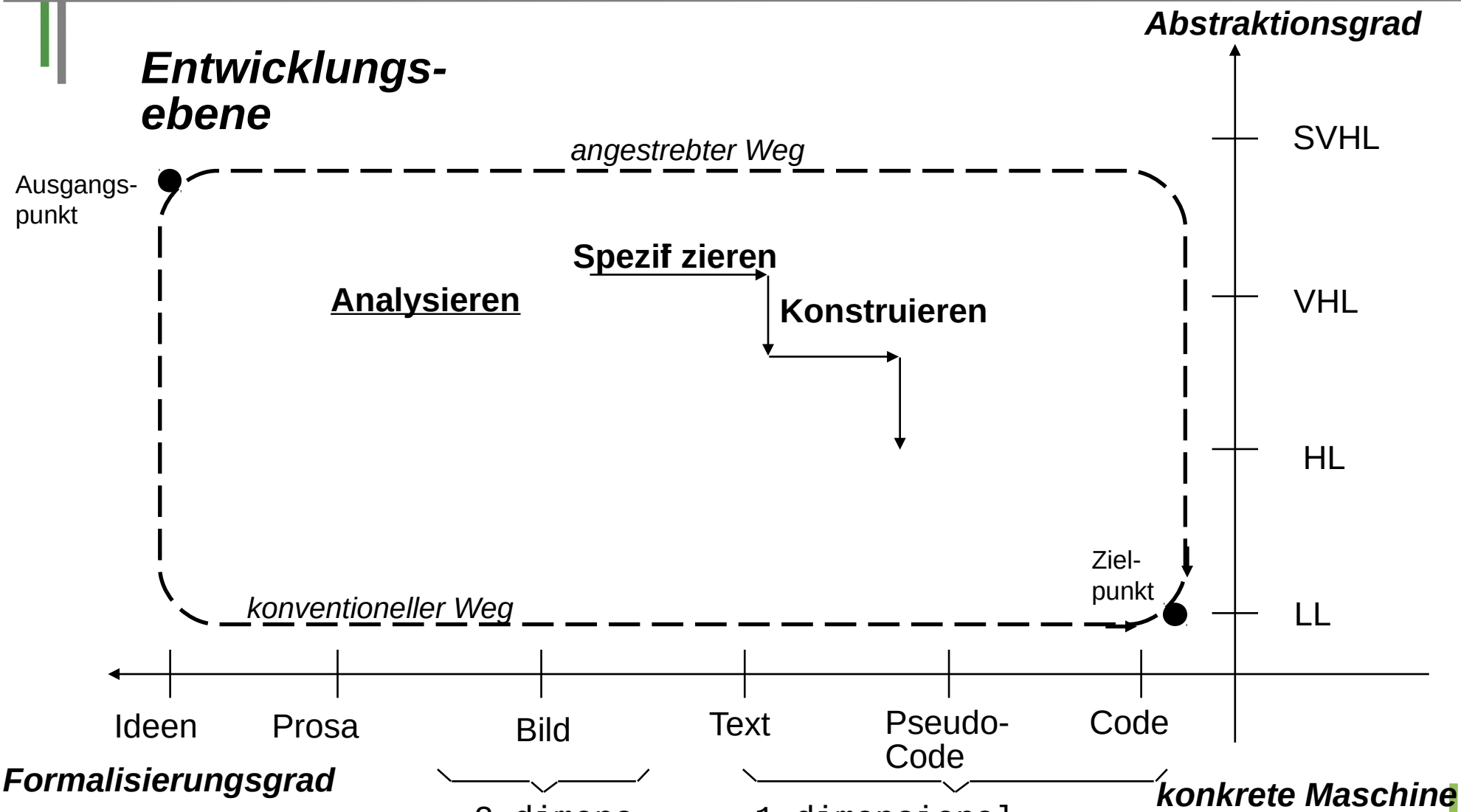
10.3 Werkzeug-Landschaft nach Hesse

Evolution der Programmiertechniken



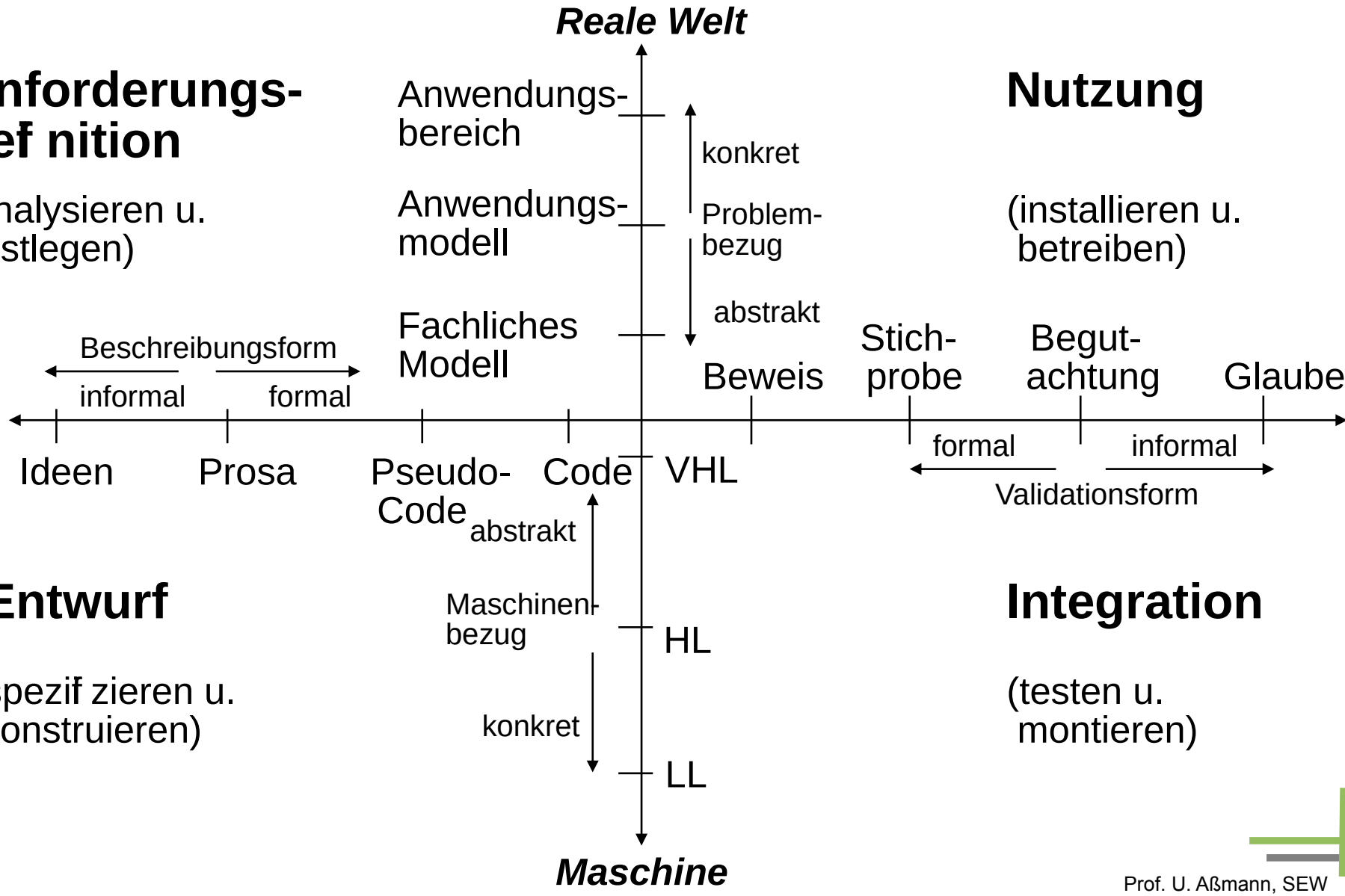
Quelle: nach [1, S. 98]

Abstraktion der Softwareentwicklung



Quelle: nach [4 HMF, S. 85]

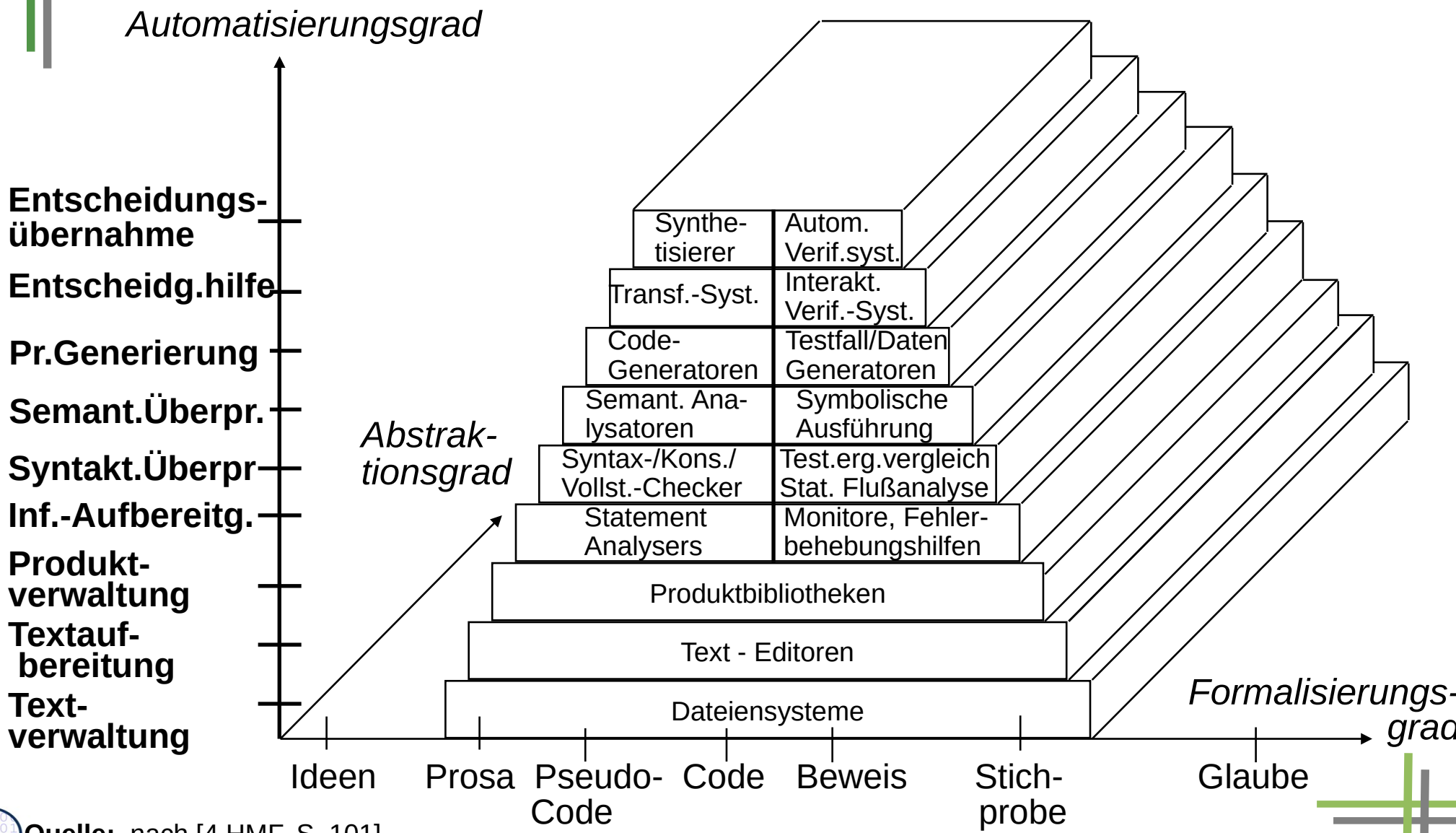
Software-Entwicklungsquadranten



Automatisierungsgrad von Werkzeugen

Nr.	Stufe	Funktion
9	Entscheidungs-übernahme	Automatisierung der Übergänge zwischen Entwicklungsschritten durch kooperierende, inferenzbasierte Werkzeuge [20]
8	Entscheidungs-Hilfe	Interaktive Transformationssysteme z.B. bei der Restrukturierung sowie bei der interaktiven Verif kation
7	(Produkt-)Gene-rierung	Automatische Erzeugung von Codegerüsten (Programmen) aus Ent-würfen und Testfällen/Testdaten aus der Anforderungsspezif kation
6	Semantische Überprüfung	Analyse z.B. des kontext-sensitiven Teils formaler Spezif kationen und andere die Programmausführung betreffende Inhalte
5	Syntaktische Überprüfung	Vollständige synt. Überprüfung formaler Spezif kationen durch „Syntax-Checker“, Parser, Flussanalysen usw.
4	Informations-Aufbereitung	Syntaktische Analyse von bestimmten formal-sprachlichen Informa-tionen, Ausgabe von Inkonsistenzen, Fehlern, Querbezügen
3	Produktverwal-tung	Manipulieren und Verwalten von wohldef nierten „Teilprodukten“, Sicherung der konsistenten Verwahrung von Versionen
2	Textaufbereitung	Fortgeschrittene Editorfunktionen, wie abschnittsweises Kopieren, Copy, Cut, Paste, Layoutfunktionen, Suchen + Ersetzen,...
1	Textverwaltung	Eingabe, Speicherung, Ausgabe von Texten mit Hilfe eines Dateisystems (normale Werkzeugfunktion)

Softwaretechnologie - Landschaft



Quelle: nach [4 HMF, S. 101]

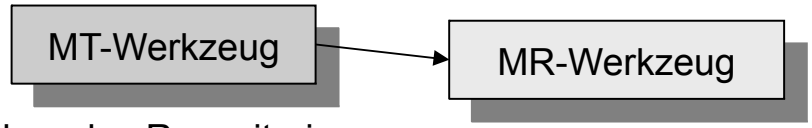
Verwendung typischer CASE-Tools

Entwicklungsphase	Spezialisierte Werkzeuge
Anforderungs-Analyse	User Interface Prototyping Tools OOA-Tools (Use Case, Object, Class) Information Modeling Tools Structured Analysis Tools Real Time Modeling Tools
Entwurf	OOD-Tools (Class Modeling) Daten-Modellierungswerkzeuge Modul/Package Specif cation Tools
Implementation Wartung	Codegeneratoren Compiler/Interpreter Symbolic Debugger Smart Text Editor Execution Prof lers Konf gurationsmanagementsysteme

Quelle: nach [8, S.22]

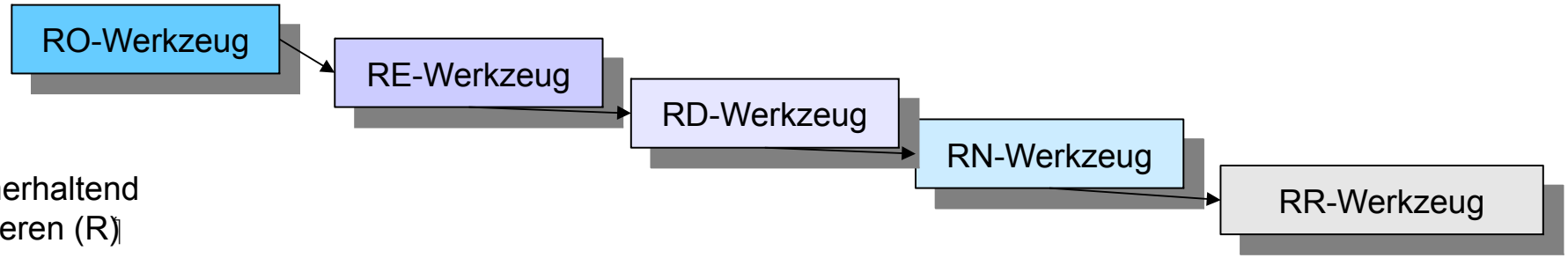
10.4 Einführung in die Effektkategorien für Werkzeuge

Effektkategorien ("Blutgruppen") für Werkzeuge



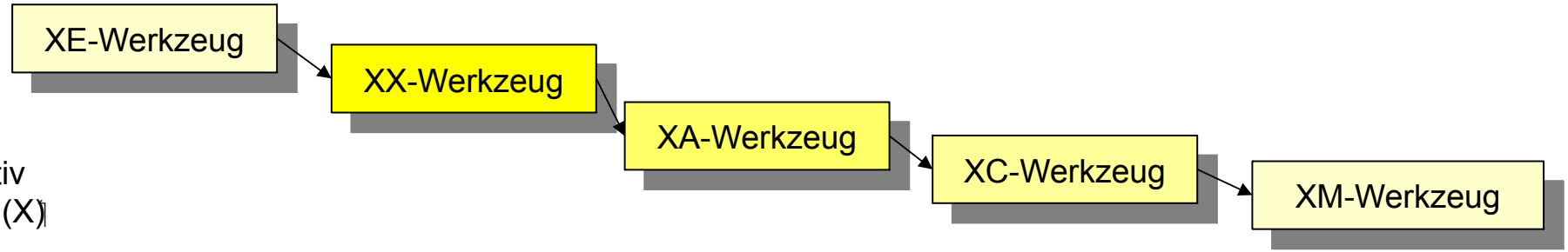
Modifizieren (M)

Modifikationswerkzeuge verändern das Repositorium



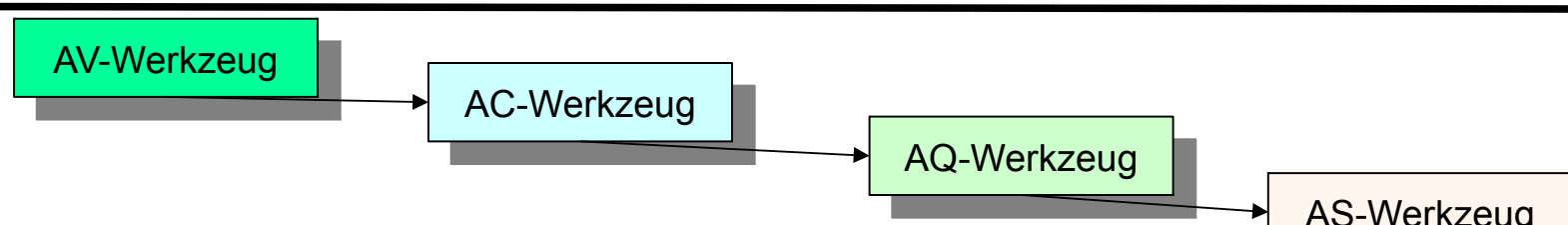
Invariantenerhaltend Restrukturieren (R)

Restrukturierende Werkzeuge verändern die Information im Repositorium, aber erhalten bestimmte Invarianten



Konservativ Erweitern (X)

Konservativ erweiternde Werkzeuge fügen dem Repositorium Informationen hinzu, zerstören aber keine Information



Analysieren (A)

Analysewerkzeuge lesen das Repositorium, verändern aber die Artefakte nicht

10.5 Der Graph-Logik-Isomorphismus

Der Graph-Logik-Isomorphismus

- Jeder Graph kann als Faktenbasis einer Logikmaschine abgelegt werden.
- Jede Faktenbasis kann als Graph interpretiert werden
  - binär: Graph
  - n-är: Hypergraph
- Logikmaschinen und Graphtransformations-Werkzeuge können zu guten Teilen ausgetauscht werden

SEU mit Ersetzungs- und Logik-Werkzeugen

Spezial-
Werkzeuge

Blutgruppe A

Logik-
werkzeuge

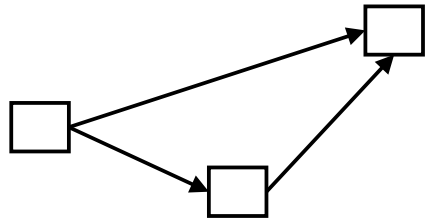
Blutgruppe M

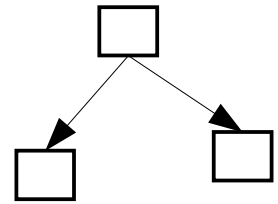
Graphersetzungs-
werkzeuge

Termersetzungs-
werkzeuge

Interpretation als Fakten

Bäume und
Graphen
Im Speicher

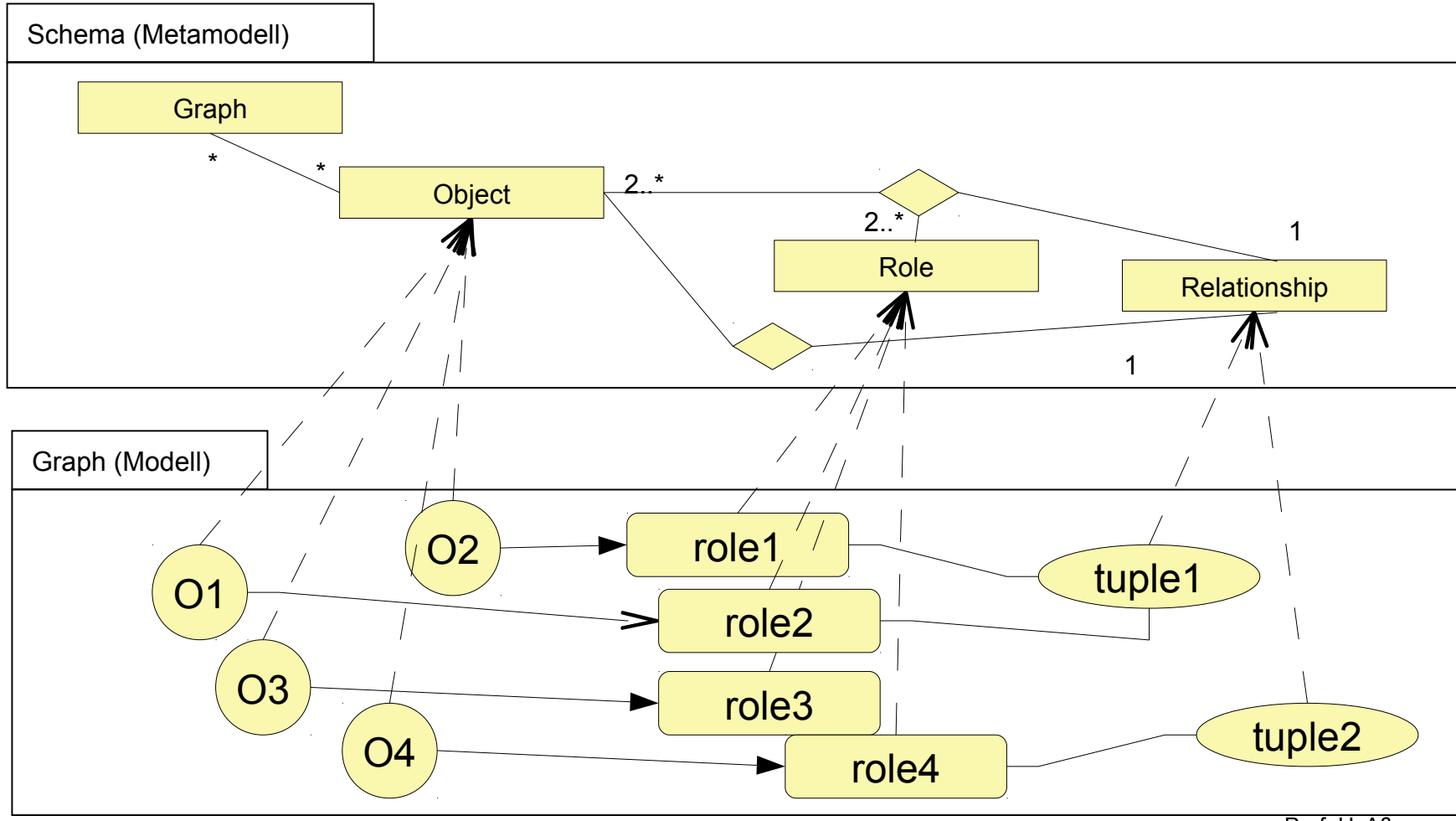




Persistente Bäume und Graphen

Typisierte Graphen (Modelle und Metamodelle)

- Graphen können typisiert sein, aber die Schemata können unterschiedlich aussehen (→ Metamodellierung)



End