Ontologien in der Softwaretechnik

Wolfgang Hesse und Barbara Krzensk

Fachbereich Mathematik und Informatik, Univ. Marburg,
Hans Meerwein-Str., D-35032 Marburg

email: hesse@informatik.uni-marburg.de, krzensk@informatik.uni-marburg.de

www: http://www.mathematik.uni-marburg.de/~hesse

Zusammenfassung:

In einigen Fachgebieten der Informatik wie der Künstlichen Intelligenz, bei Datenbank- und Web-basierten Systemen spielt der Ontologiebegriff eine zunehmend wichtige Rolle - er steht dort (anders als in der Philosophie) für die explizite formale, projektübergreifende Konzeptualisierung eines Anwendungsbereichs. Für die Softwaretechnik sind Ontologien hauptsächlich in den frühen Phasen von Software-Projekten interessant, um Modelle und Anwendungs-Komponenten besser standardisieren und wieder verwenden zu können

In diesem Beitrag werden konzeptuelle Modelle in Software-Projekten und Ontologien sowie die jeweiligen Vorgehensweisen bei deren Entwicklung einander gegenübergestellt sowie die Grundzüge eines *Ontologie-basierten Software-Engineering (OBSE)* skizziert.

Abstract:

The term *ontology* is becoming increasingly important in several fields of Informatics like Artificial Intelligence, Database or Web Technology. Unlike its originnal meaning in Philosophy *ontology* stands for *a formal explicit specification of a shared conceptualization.* In the Software Engineering field ontologies play a major role in the early phases of software projects since they might facilitate the reuse of models and standardized application components.

In this contribution, conceptual models of software projects are contrasted with ontologies and the corresponding life cycle models for their development are compared. Finally, principles of an *Ontology-based Software Engineering (OBSE)* approach are sketched.

1 Begriffsbestimmung - Was ist eine Ontologie?

Ehe man sich über Ontologie (oder Ontologien) im Bereich der Informatik unterhält, ist es notwendig, diese Neuverwendung eines alten Begriffs gegenüber seiner überlieferten Verwendung im Bereich der Philosophie abzugrenzen. Dort steht Ontologie für die *Lehre vom Sein* - genauer von den Bedingungen und Möglichkeiten des *Seienden* - und ist eng verwandt mit der *Erkenntnistheorie*, die sich mit den Möglichkeiten und Grenzen menschlichen Wahrnehmens und Erkennens auseinandersetzt.

In der Informatik - das ist der erste und entscheidende Unterschied - nimmt man das "Seiende" als gegeben hin, grundsätzliche Erwägungen über dessen Ursprünge oder Bedingungen

spielen kaum eine Rolle. Anders steht es mit der Erkenntnisfrage: Ob man nun die Welt als Realist oder Idealist (oder als ein sonstiger …-ist) betrachtet: man kann nie sicher sein, was und wie sie nun wirklich ist - jede Weltbeschreibung und Weltdeutung muss notwendigerweise unvollständig, eingeschränkt, selektiv sein, ist immer von bestimmten Zielen und Zwecken geleitet und damit hinterfragbar. Außerdem hängt sie stark von den zur Verfügung stehenden Beschreibungsmitteln und deren Verwendung ab, ist also von Sichtweise und Sprache bestimmt.

Hier setzt die Informatik als Sprach- und Kommunikationswissenschaft an: Wenn immer Menschen sich - mit oder ohne Computer - erfolgreich über Bereiche der Lebenswelt und die dort angesiedelten Dinge und Vorgänge austauschen wollen, ist ein gemeinsames Begriffs- und Strukturverständnis dieses Bereichs unabdingbare Voraussetzung dafür. Soll ein solcher Austausch gar zwischen Computerprogrammen (ohne menschliche Interaktion) möglich sein, so sind die Anforderungen an die Eindeutigkeit und Rigidität solcher Beschreibungen umso höher. Damit ist klar, worauf jegliche Beschäftigung mit "Ontologie" in der Informatik abzielt:

"Ontologies provide a shared and common understanding of a domain that can be communicated between people and application systems." (Hensel [Hen 00]

"The role of ontologies is to capture domain knowledge in a generic way and to provide a commonly agreed upon understanding of a domain. The common vocabulary of an ontology, defining the meaning of terms and their relations, is usually organised in a taxonomy and contains modelling primitives such as classes, relations, functions, and axioms." [MSS+ 00]

Oder - in der bekanntesten und kurz gefassten Definition von T. Gruber: [Gru 93]):

"An ontology is a formal explicit specification of a shared conceptualization"

Damit ist weiter klar, dass die Rede von "Ontologien" (im Plural) in der Informatik - im Gegensatz zur Philosophie - durchaus Sinn macht, denn erstens gibt es beliebig viele durch Ontologien beschreibbare Lebensbereiche und zweitens gibt es zu jedem dieser Bereiche beliebig viele Sichtweisen und sprachliche Ausdrucksformen, die (zumindest prinzipiell) zu beliebig vielen Ontologien führen können. Ob diese Koexistenz vom Standpunkt der Verständigung wünschbar oder tolerierbar ist, steht auf einem ganz anderen Blatt.

Die Verwendung des Wortes "Ontologie" in der Informatik ist nicht unumstritten - Philosophen können mit Recht darin einen Missbrauch sehen [Jan 01]. In der Tat könnten die Informatiker statt "Ontologie" *standardisiertes Referenzmodell für einen bestimmten Anwendungsbereich* sagen - aber wäre das praktischer? Kürzer wäre es jedenfalls nicht - man müsste also zumindest ein anderes Kunstwort dafür erfinden. So lange wir das nicht haben, mag es bei "Ontologie" bleiben - die Gefahr der Missdeutung scheint uns jedenfalls relativ gering zu sein.

2 Ontologien in der und für die Softwaretechnik

Nach dem oben Gesagten ist weiterhin klar, welche Bereiche in der Informatik hier vorrangig angesprochen sind: Künstliche Intelligenz (KI), Datenbanken, Internet-Technologie. Welche Rolle können (und sollten?) Ontologien für die Softwaretechnik spielen? Wir sehen prinzipiell zwei Ansatzpunkte:

- *Ontologien in der Softwaretechnik:* Wiederverwendung ist ein zentrales Thema der Softwaretechnik seit deren Anfängen. Lange Zeit konzentrierte man sich auf den mehr-

fachen Einsatz fertiger Software-Komponenten oder -Module. Diese Form der Wiederverwendung muss sich zwangsläufig auf systemnahe, vorwiegend anwendungs-neutrale Komponenten beschränken, für anwendungs-spezifische stellt sie eher einen Ausnahmefall dar. Fragen nach der standardisierten Beschreibung von Anwendungsbereichen stellen sich hier kaum.

Anders liegt der Fall, wenn man die Idee der Wiederverwendung auf *konzeptuelle Modelle* ausdehnt. Im Zeitalter der zunehmend zusammenwachsenden Anwendungslandschaft(en) steckt hier ein großes Potenzial für die Entwicklung, Pflege und Nutzung von Ontologien.

- *Ontologie(n) für die Softwaretechnik:* Die Softwaretechnik ist selbst ein eigenständiges Fachgebiet der Informatik mit einer eigenen Struktur und Terminologie. Da es sich zudem um ein relativ junges Gebiet handelt, wäre hier die Entwicklung einer (oder mehrerer ?) Ontologie(n) eine besonders herausfordernde Aufgabe. Ansätze dazu gibt es einige, z. B. Begriffsgerüste [GI 03] oder das SWEBoK-Projekt [DBA 99].

3 Sind Ontologien konzeptuelle Modelle - oder was unterscheidet sie?

Im Folgenden wollen wir uns auf den ersten Punkt: *Ontologien in der Softwaretechnik* konzentrieren. Beim ersten Hinsehen sieht es so aus, als ob Ontologien in der Softwaretechnik nichts anderes sind als die seit P. Chen dort wohl bekannten und reichlich genutzten konzeptuellen Modelle (KM) - häufig repräsentiert durch Entity/Relationship (E/R-) Diagramme. In der Softwaretechnik konnten diese nutzbringend bei der Datenmodellierung, d.h. bei der Festlegung und Darstellung der wesentlichen für ein Projekt benötigten Datenstrukturen und deren Zusammenhängen eingesetzt werden. E/R-Diagramme wurden in vielen Varianten neu- und umdefiniert und durch zusätzliche Konstrukte (z.B. Vererbungs-Beziehungen) angereichert.

Mit dem Aufkommen der objektorientierten (OO-) Modellierungstechniken und vor allem der Unified Modelling Language (vgl. [UML 01]) kamen Aspekte der funktionalen Modellierung hinzu, an die Stelle der E/R-Diagramme traten Klassendiagramme, die (teilweise) auch Operationsdefinitionen enthielten. Wir wollen auch diese zu den Techniken der konzeptuellen Modellierung rechnen.

Ebenso wie die genannten Techniken zielen Ontologien auf die zusammenhängende Darstellung der relevanten "Dinge" eines Anwendungsbereichs und deren Bezüge untereinander. D.h. grundsätzlich können sich Ontologien der gleichen Beschreibungsmittel und Darstellungstechniken bedienen. Ebenso wie bei den KM können wir einen "syntaktischen" und einen "semantischen" Bereich unterscheiden: Der erstgenante umfasst die Benennung, Klassifizierung und strukturelle Beschreibung von "Dingen" (*entities*) und Bezügen (*relationships*), ggf. auch der von diesen ausführbaren Operationen mit Namen, Parameter- und Ergebnistypen. Zur "Semantik" gehören weitergehende, durch die Diagramme oder Formalsprache i.a. nicht unmittelbar ausdrückbare Bedingungen und Zusammenhänge. Ein KM ist umso "semantischer", je mehr es gelingt, solche Phänomene zu formalisieren.

So weit zu den Gemeinsamkeiten. Es gibt aber auch Unterschiede: Sie fallen spätestens dann auf, wenn wir die unterschiedlichen Zielsetzungen miteinander vergleichen. Herkömmliche KMe sind i.a. auf *ein* Projekt (oder maximal auf eine vorgegebene, begrenzte Projektfamilie) bezogen, sie erheben keinen Anspruch auf Gültigkeit über die Projektgrenzen hinaus. Daher sind auch die Anforderungen an ihre Verbindlichkeit und Stabilität auf das Projekt bzw. die Projektfamilie begrenzt: Ein KM kann immer dann geändert werden, wenn die von der Änderung betroffenen Projektmitarbeiter informiert sind und dem zustimmen.

Anders verhält es sich im Falle der Ontologie: Der Kreis derjenigen, die das dort festgelegte Wissen teilen (im Sinne des *sharing* von Gruber's Definition, vgl. oben) ist ungleich größer. Er macht an Projektgrenzen nicht halt und schließt künftige Projekte und Entwicklungen für den gleichen Anwendungsbereich sowie potenzielle, womöglich noch nicht bekannte Teilhaber mit ein. Der vergrößerte Kreis derjenigen, von denen ein *ontological commitment* erwartet und benötigt wird, macht die Definition und etwaige Änderungen ungleich schwerer.

Spyns et al. stellen in [PMJ 02] wesentliche Eigenschaften von Datenmodellen und Ontologien einander gegenüber:

- *Ziel und Zweck:* Datenmodelle sollen zwischen den Entwicklern einer bestimmten Anwendung untereinander und deren Nutzern vermitteln. Ontologien beschreiben dagegen Wissen, das allen in einen bestimmten Anwendungsbereich involvierten Personen (Angehörigen von Fachabteilungen, Herstellern von Werkzeugen, Systementwicklern, Endbenutzern, Zulieferern, etc.) gemeinsam ist. Entscheidungen über die Granularität eines Modells oder die Wahl zwischen Klasse oder Attribut als Modellelement sind im ersten Fall vom Projektziel bestimmt, im zweiten dagegen von den Bedürfnissen vieler, möglicherweise inhomogen zusammengesetzter Benutzergruppen.

- *Ausdruckskraft* der Beschreibung: Regeln beziehen sich bei Datenmodellen auf die Integrität der Daten und auf Einzelheiten der Implementierung wie z.B. Schlüsselwerte, Fremdschlüssel, Nullwerte, … Bei Ontologen sind sie dagegen allgemeingültig, beziehen sich in erster Linie auf die Konzeptualisierung und müssen implementierungs-unabhängig sein. Entsprechend unterschiedlich sind die Anforderungen an die Beschreibungsmittel: detailliert, darstellungs-bezogen im ersten Fall, generalisierend, möglichst darstellungsunabhängig, Ableitungen unterstützend im zweiten Fall.

- *Erweiterbarkeit:* Die diesbezüglichen Anforderungen sind für Ontologien ungleich höher als für Datenmodelle. Bei den letzteren sind mögliche Erweiterungen abzusehen, sie beschränken sich auf das aktuelle Projekt oder eine klar abgegrenzte Projektfamilie. Eine Ontologie muss dagegen aufgrund ihrer Bestimmung immer offen für neue Projekte und Anwendungen mit neuen Anforderungen an mögliche Erweiterungen sein.

Die genannten Autoren verweisen (mit Recht) auf das Dilemma zwischen Generizität und Effektivität: Je allgemeiner eine Ontologie gefasst ist, desto breiter ist sie verwendbar - aber desto unspezifischer und (im konkreten Fall) weniger effektiv einsetzbar wird sie. Entsprechend gilt das Umgekehrte: je spezifischer die Ontologie, desto eingeschränkter ihr Anwendungsbereich.

4 Grundforderungen an Ontologien im Software-Entwicklungsprozess

Welche Forderungen ergeben sich für den Software-Entwicklungsprozess, die darin definierten Aktivitäten und zu erbringenden Resultate, wenn man die Idee der Ontologie-Entwicklung und -Nutzung in diesen einbringen wollte? Natürlich sind in erster Linie alle diejenigen Phasen betroffen, in denen die Anwendung eine besondere Rolle spielt - d.h. die ganz frühen Phasen der *System- und Anforderungsanalyse* sowie die ganz späten der *Installation, Inbetriebnahme, Nutzung und Revision* der erstellten Software.

Nach dem heute propagierten und vorrangig praktizierten, UML-unterstützten Vorgehen steht am Beginn der Software-Entwicklung eine *Analyse* von *Anwendungsfällen* (*use case analysis*). Diese findet hauptsächlich auf der verbalen Ebene statt, wird kaum durch formale Verfahren unterstützt und führt zu einem oder mehreren (i.a. wenig aussagekräftigen) Anwen-

dungsfall-Diagramm(en) und einer Reihe von funktionalen, in natürlicher Sprache gehaltenen Beschreibungen der einzelnen Anwendungsfälle (vgl. z.B. [JBR 99]). Die darin vorkommenden Objekte, ihre Eigenschaften, Zustände etc. werden durch frei gewählte Bezeichner referenziert, ihre Auswahl, Vollständigkeit, Konsistenz unterliegt keiner formalen Kontrolle. Jacobson spricht z.B. von einer *Objektliste*, auf deren Grundlage dann das so genannte Analysemodell, d.h. ein oder mehrere UML-Klassendiagramme zu erstellen ist bzw. sind.

Ein solches - schon für "normale" Entwicklungsprojekte recht dürftiges - Vorgehen erweist sich für Ontologie-basierte Projekte als unbrauchbar. Das Sammeln, Entwerfen und Dokumentieren von Anwendungsfällen oder ähnlich gearteten Geschäftsprozess-Beschreibungen erfordert eine intensive Interaktion mit den für den Anwendungsbereich relevanten Gegenständen, Vorgängen und ihren Beziehungen - also mit der dafür zuständigen Ontologie. Dort sollten alle relevanten Begriffe für vorkommende Objekte, Eigenschaften, Beziehungen, Zustände etc. verzeichnet und terminologisch festgelegt sein. Nicht verzeichnete Begriffe müssen der Ontologie zugefügt werden, so weit sie von allgemeinem Interesse sein könnten. Dazu sind Ontologie-Erweiterungen und möglicherweise -Modifikationen notwendig. Diese erfordern eine enge Zusammenarbeit mit der für die Ontologie-Entwicklung zuständigen Institution, z.B. einem Gremium von Anwender- und Herstellervertretern.

In welcher Form können ontologische Festlegungen niedergelegt und anwendergerecht kommuniziert werden? Ist ein Analysemodell mit UML-Klassendiagrammen dafür geeignet? Die Antwort lautet *nein*, weil der Anspruch auf Verallgemeinerbarkeit weitgehend verletzt wäre. Können Entscheidungen, ob ein Anwendungs-Gegenstand als Klasse, als Attribut oder als Operation zu modellieren sind, ontologie-weit getroffen werden? Wiederum *nein*, weil solche Entscheidungen offenkundig projektspezifisch, abhängig von den Zielen der konkreten Anwendungsentwicklung getroffen werden müssen.

Einige Autoren plädieren daher für einen *Glossar-basierten* Ansatz, bei dem alle Ontologie-Elemente in einem Glossar definiert, ihre Beziehungen verbindlich benannt und durch Regeln festgelegt werden. Beim KCPM-Ansatz wird dies teilweise auch durch Werkzeuge unterstützt [M-K 02].

Spyns et al. verfolgen einen solchen Ansatz und stützen sich dabei auf das ORM-Paradigma (*Objects with Roles-Model*, [Hal 01]). Die Autoren unterscheiden grundsätzlich zwischen einer allgemein verbindlichen *Ontologie-Basis (ontology base)* und Paketen von bereichsspezifischen Regeln (*domain rules*), die sie als *ontological commitments* bezeichnen. Ein solches *commitment* spielt die Mittlerrolle zwischen der Ontologie-Basis und ihren Anwendungen. In seinen Regeln wird festgelegt, welche Teile der Ontologie für die spezifische Anwendung sichtbar sind und mit welchen zusätzlichen Bedingungen und Einschränkungen sie belegt sind [SMJ 02].

Das Aufbauen eines Glossars und damit die Arbeit mit einer Ontologie besteht also immer aus zwei Teilen:

- der Analyse und Bearbeitung der Ontologie-Basis und
- der Adaption dieser Basis an das eigene Projekt und die eigene Anwendung durch ontologisches *commitment*, d.h. Festlegung von Regeln für deren Gebrauch, Einschränkung und Ausdehnung.

Dabei besteht für Spyns et al. die Ontologie-Basis aus einer Sammlung von Fakten-Deklarationen (sog. *Lexons*, vgl. [SMJ 02]). Diese kann man sich etwa in der folgenden (modifizierten und verallgemeinerten) Form vorstellen:

<Kontext-Id> :: <Term>

bzw:

<Kontext-Id>.<n>-ary :: <Term> <$Term_1$> .. <$Term_n$> , wobei $n \in \mathbb{N}^+$

Dabei steht jeder *Term* für eine (n-stellige) Assoziation. Null-stellige Assoziationen können als einfache, nicht weiter strukturierte Dinge, einstellige als Beschreibung von Dingen durch Eigenschaften gedeutet werden. Jeder Term kann (optional) durch eine in Klammern eingeschlossene Typ- oder Wertangabe parametrisiert werden. Fakten können auf der Typ- oder Exemplar-Ebene formuliert werden oder diese miteinander verbinden.

Ein *commitment* besteht nach [SMJ 02] aus einer Menge von halbformalen - d.h. teils in natürlicher Sprache, teils in der Formalsprache RIDL - formulierten Regeln. Im Verlauf eines Software-Projekts könnten diese Regeln - entsprechend dem Entwicklungsstand des Projekts - in mehreren Versionen auftreten: zunächst vorwiegend informal, im Lauf der fortschreitenden Präzisierung und Implementierung zunehmend formaler.

Zusammenfassend wollen wir festhalten:

- Eine Ontologie kann den Software-Entwicklungsprozess vereinfachen und die entstehenden Systeme vereinheitlichen - vor allem im Hinblick auf mögliche Interoperabilität und Wiederverwendung. Die Ontologie sollte Anwendungswissen so weitgehend und akkurat wie möglich wiedergeben, aber strukturell so wenig wie möglich Vorentscheidungen treffen, die auf die Implementierung Einfluss haben könnten.

5 Der Ontologie-Lebenszyklus

Ehe wir uns der Frage der Interaktion von Software- und Ontologie-Entwicklungsprozessen zuwenden, wollen wir den Ontologie-Lebenszyklus für sich allein betrachten. Frühere Autoren haben auf eine gewisse Analogie zum Software-Entwicklungsprozess hingewiesen und untersuchen daher bekannte Vorgehensmodelle als mögliche Vorbilder. Grundsätzliche mögliche Vorgehensweisen sind z.B. (vgl. [FGJ 97]):

- Wasserfall-Modell,
- inkrementeller Ansatz,
- evolutionärer Ansatz.

Es zeigt sich schnell, dass Wasserfall-artige Modelle ungeeignet sind. Ontologie-Entwicklung verläuft in der Regel nicht sequentiell. Besser geeignet ist ein inkrementeller Ansatz, aber auch dieser entspricht oft nicht der Realität: Eine Ontologie wird i.a. nicht als Folge von Inkrementen aufgebaut. Dagegen passt ein evolutionärer Ansatz gut: Eine Ontologie wird zunächst "prototypisch" formuliert und muss dann laufend - zusammen mit Projekten, die für den Anwendungsbereich relevant sind, angepasst und weiterentwickelt werden. Jedes relevante Projekt stößt also im Prinzip einen neuen Ontologie-(Weiter-)Entwicklungszyklus an.

Unser Ansatz folgt dem EOS-Prinzip (vgl. [Hes 96], [Hes 03]):

- Komponentenweiser Aufbau: Eine Ontologie kann in mehrere Unter-Ontologien aufgeteilt werden, diese müssen kompatibel sein und miteinander integriert werden können. Jede Ontologie (außer der "top level"-O.) ist Teil einer übergeordneten Ontologie und muss mit dieser integriert werden können.

- Entwicklungszyklen: Jede Ontologie (-Komponente) hat ihren eigenen, i.a. beliebig oft wiederholten Entwicklungszyklus bestehend aus den Phasen:

. Analyse
. Entwurf
. Implementierung
. Einsatz und Bewertung

Es folgen einige kurze Erläuterungen zu den einzelnen Phasen:

- **Analyse:**

. Ziel und Zweck (*purpose*) feststellen: Wozu wird die Ontologie entwickelt?

. Untersuchungsbereich festlegen, Gegenstandsbereich (*scope*) abgrenzen: Worauf bezieht sich die Ontologie?

. Einbettung und Schnittstellen festlegen: Welche schon vorhandenen oder entstehenden Ontologien sind relevant?

. Bezüge zu relevanten Ontologien: Klassifizieren in übergeordnete / angrenzende / konkurrierende / potenzielle Vererber- Ontologien; Unterschiede, Konflikte identifizieren.

. Anwendungsfälle ("Use cases"): sind hier existierende oder projektierte Anwendungen, d.h. alle Projekte, die die Ontologie nutzen (könnten)

. Wissens-Quellen: Woher ist das Wissen für die Ontologie zu akquirieren?

. Formalität, Beschreibungsmittel klären

. "Objektliste" (Basis für Ontologie-Basis) erstellen. Ergebnis: Glossar - z.B. im Sinne von Spyns et al. [SMJ 02] bzw. von KPCM, vgl. Mayr/Kop, [M-K 02])

. Strategie zur Ontologie- (Weiter-) Entwicklung und -Integration klären

- **Entwurf:**

. Hierarchie und Struktur festlegen: ggf. Aufteilung in Unter-Ontologien: Abgrenzung, Schnittstellen festlegen.

. (ggf.) eigene (Unter-) Entwicklungszyklen starten (vgl. EOS-Modell)

. Konzeptualisierung des aktuellen, zur Debatte stehenden Bereichs. Ergebnis: Zusammenhängendes Glossar mit passenden Querbezügen und Erklärungen. Mögliche Übersetzungen in "Dialekte" wie E/R- oder Klassendiagramm(e), Topic Maps o.ä.

. Formalisierung: Definition der Regeln in einer formalen Sprache.

. Prüfungen auf Konsistenz und Vollständigkeit

- **Implementierung:**

. Umsetzung in eine konkrete Ontologie- oder Programmiersprache, z.B. DAML+OIL, OWL, XML+RDF, DL, Prolog, Java-Klassen

. (Innere) Integration von ggf. gebildeten Teil-Ontologien

- **Einsatz und Bewertung:**

. ggf. (äußere) Integration mit umgebenden / übergreifenden Ontologien (s. dazu [FGJ 97], S. 2)

. Prüfen auf Überlappungen, Inkonsistenzen mit umgebenden / übergreifenden Ontologien

. Entgegennehmen von Rückmeldungen von verwendenden Projekten / von Anwendern / Fachabteilungen

. Evaluierung: auf Konsistenz, Vollständigkeit, Angemessenheit

. Revision

In der folgenden Tabellen wollen wir die Gemeinsamkeiten und Unterschiede der Vorgehensmodelle einander gegenüberstellen:

	Software Engineering	**Ontology Engineering**
Prozess	• Prozess-Dauer - fest definiert, auf Projekt begrenzt • Prozess-Struktur - Phasen, evtl. in Iterationen unterteilt (z.B. im RUP) - Aktivitäten: in Phasen eingeordnet • Subprozesse beziehen sich auf die Entwicklung einzelner Komponenten oder Inkremente • Prozess-Paradigmen: - Wasserfall - inkrementell - Komponenten-basiert - prototypisch - evolutionär	• Prozess-Dauer - unbestimmt, unbegrenzt • Prozess-Struktur - vorrangig Iterationen, ggf. in Phasen unterteilt - Aktivitäten: i.W. Ausbau, Modifikation von Teilbereichen • Subprozesse beziehen sich auf die Entwicklung oder Revision einzelner Teilbereiche • Prozess-Paradigmen: - inkrementell - Komponenten-basiert - evolutionär
Analyse	• Tätigkeiten: - Anforderungen erheben - Anwendungsfälle (use cases)	• Tätigkeiten: - potenzielle Anwendungen identifizierten

	analysieren - Objektlisten bilden - Klassen-Grobstruktur entwerfen • Ergebnisse: - Anforderungen - Anwendungsfälle (use cases) + Anw.-fall-Diagramm - Objektlisten - Klassenmodell (Vorentwurf) • Beschreibungsmittel: - Anw.-fall-Diagramm - Klassendiagramme, UML - nat. Sprache - halbformal - vorrangig an Entwickler und Anwender gerichtet	- Terminologie sichten und ordnen - Glossar anlegen - Fakten und Regeln sammeln - Terminolog. Konflikte erkennen • Ergebnisse: - (potenz.) Anwendungen - Glossar(e) - Fakten und Regeln • Beschreibungsmittel: - nat. Sprache - Tabellen - Semantische Netze - in- oder halbformal - vorrangig an Bereichs-Experten gerichtet
Entwurf	• Tätigkeiten: - Klassen-Feinstruktur entwerfen - System-Architektur (Komponenten- und Modulstruktur) entwerfen - Bausteine spezifizieren • Ergebnis: - Konzeptuelles Modell - Struktur E-R-A (Chen et al.) • Beschreibungsmittel: - E-R-Diagramme - Klassendiagramme, UML - halbformal - vorrangig an Entwickler und Anwender gerichtet	• Tätigkeiten: - Struktur festlegen (ggf. Unter-Ontologien) - Glossar(e) füllen, abgleichen - Terminolog. Konflikte lösen • Ergebnisse: - Ontologie-Struktur - gefüllte Glossar(e) - Fakten und Regeln - Struktur: Fakten (O-R, z.B. Nijssen, ORM) • Beschreibungsmittel: - nat. Sprache - Logik-Sprachen - Semantische Netze - informal / halbformal, - vorrangig an Bereichs-Experten gerichtet
Implementierung und Integration	• Tätigkeiten: - codieren: Umsetzen in konkrete Programmiersprache(n) - testen, Fehler beheben - integrieren, Bausteine zusammenfügen und gemeinsam testen • Ergebnis: - getestete Programme	• Tätigkeiten: - formalisieren: Umsetzen in formale Ont.-Sprache(n) - formale (syntakt. und semnat. Prüfungen durchführen) - integrieren: Teil-Ont. zusammenfügen und abgleichen • Ergebnis: - formaler Fakten- & Regelsatz

	- integriertes System • Beschreibungsmittel: - Programmiersprache(n)	- formaler Fakten- & Regelsatz • Beschreibungsmittel: - Logik-Sprachen - Semantische Netze - Frames, DAML+OIL. OWL - formal - vorrangig an Austausch mit anderen Programmen gerichtet
Einsatz und Bewertung	• Tätigkeiten: - System beim Anwender installieren und einsetzen - prüfen, bewerten, Rückmeldungen sammeln - Fehler beheben, Anforderungen an Revision sammeln - (ggf.) Revision anstoßen • Ergebnis: - Prüfberichte - Revisions-Anforderungen	• Tätigkeiten: - Ont. veröffentlichen - prüfen, bewerten, Rückmeldungen der Anwender aufnehmen - Ont. an Nachbar-Ont. anpassen, Inkonsistenzen beheben - Anforderungen an Revision sammeln - (ggf.) Revision anstoßen • Ergebnis: - Anwenderberichte - Revisions-Anforderungen
Ergebnisse/ Produkte	- projekt-bezogen - i.a. nicht weiterverwendet - (relativ) kurzlebig - isoliert, nur für den Projekt-Bereich gedacht	- projekt-übergreifend - wieder verwendbar - langlebig - "sharable": für viele Institutionen und Projekte bindend

6 Ausblick: Wie könnte ein *Ontologie-basiertes Software Engineering* aussehen?

Zum Abschluss wollen wir kurz unsere Ideen zu einem *Ontologie-basierten Software Engineering (OBSE)* in Form einiger knapper Thesen skizzieren.

- Ontologie-Entwicklung und Software-Entwicklung verlaufen in parallelen, aber unter schiedlichen Zyklen mit verschiedenen Zielsetzungen, Verantwortlichkeiten und Zeithorizonten. Ontologie-Entwicklung ist langfristiger und breiter angelegt, betrifft i.a. viele Software-Projekte und obliegt oft heterogen zusammengesetzten Gremien.

- Software-Projekte, die (z.B. aus den eingangs genannten Gründen) ontologie-basiert ablaufen sollen, können sich in ihren frühen Phasen an die/eine Ontologie-Entwicklung des betreffenden Anwendungsbereichs "andocken" und nach Abschluss ihre Ergebnisse in diese einbringen. Als gemeinsames, evolutionäres Entwicklungsmodell nutzen wir das EOS-Modell (vgl. oben und Abb. 1).

- Eine Ontologie kann zunächst als eine spezielle *Komponente* in der Software-Entwicklung aufgefasst werden (s. Abb.1) . Das darin niedergelegte Wissen geht als *"codifiziertes Modell"* in die projektspezifische Modellbildung ein und spiegelt sich dann womöglich in mehreren Komponenten des Software-Systems wider.

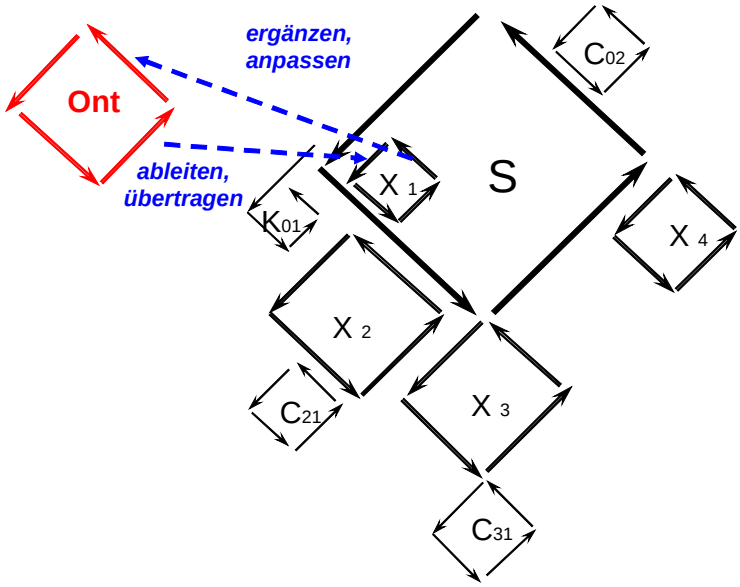

Abb. 1: Ontologie- und Software-Entwicklungszyklen im EOS-Modell

- Ontologie- und Software-Entwicklung sind miteinander verzahnt und greifen in folgender Weise ineinander:

  - System-Analyse (SA) impliziert das Ableiten und Übertragen von ontologischen Festlegungen für das zur Debatte stehende Anwendungsgebiet ,

  - System-Entwurf (SE) basiert auf ontologischen Definitionen und *commitments* für das spezifische Projekt,

  - System-Implementierung und operationeller Einsatz (SI/SO) können dazu führen, dass Projekt-Ergebnisse und -Erfahrungen in die Ontologie-(Weiter-) Entwicklung eingebracht und dort nutzbar gemacht werden .

- Insgesamt sorgen Ontologien für den *Wissenstransfer* von Projekt zu Projekt and von einem Entwicklungszyklus für ein bestimmtes Anwendungsgebiet zum nächsten. Sie erhalten damit eine zentrale Bedeutung als Wissensbanken für Software- Anwendungsbereiche.

Literaturhinweise:

[DBA 99] R. Dupuis, P. Bourque, A. Abran, J. W. Moore, and L. L. Tripp :The SWEBOK Project: Guide to the Software Engineering Body of Knowledge Presented at ICSSEA '99 Paris, France December 8-10, 1999.

[FGJ 97] M. Fernandez, A. Gomez-Perez, N. Juristo: METHONTOLOGY: From ontological art towards ontological engineering, Symp. on Ontological Engineering of AAAI, Stanford Ca. (1997)

[Fer 99] Fernández-Lopez, M.: Overview of Methodologies For Building Ontologies, Proc. IJCAI-99 Workshop on Ontologies and Problem-Solving Methods, Stockholm 1999

[Gua 98] N. Guarino: Formal Ontology and Information Systems. In: Proc. FOIS '98, Trento (Italy) June 1998, Amsterdam IOS Press pp 3-15

[GI 03] Gesellschaft für Informatik (GI)-Arbeitskreise "Terminologie der Softwaretechnik" und "Begriffe für Vorgehensmodelle": Informatik-Begriffsnetz. http://www.tfh-berlin.de/%7Egiak/ (Stand v. 1.12.2003)

[Gru 93] T. Gruber: A translation approach to portable ontologies. Knowledge Acquisition, 5(2), pp. 199-220 (1993)

[G-L 02] M. Gruninger, J. Lee: Ontology - Applications and Design. CACM 45.2, pp. 39-41 (Feb. 2002)

[G-W 02] N. Guarino, Ch. Welty: Evaluating Ontological Decisions with Ontoclean. CACM 45.2, pp. 61-64 (Feb. 2002)

[Hal 01] T. Halpin: Information Modeling and Relational Databases: from conceptual analysis to logical design. Morgan-Kaufmann 2001

[Hen 00] D. Hensel: *Relating Ontology Languages and Web Standards*. In: J. Ebert, U. Frank (Hrsg.): *Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik. Proc. "Modellierung 2000"*, Fölbach-Verlag, Koblenz 2000, pp. 111-128

[Hes 96] W. Hesse: Theory and practice of the software process - a field study and its implications for project management; in: C. Montangero (Ed.): Software Process Technology, 5th European Workshop, EWSPT 96, Springer LNCS 1149, pp. 241-256 (1996)

[Hes 02] W. Hesse: Das aktuelle Schlagwort: Ontologie(n). in: Informatik Spektrum, Band 25.6 (Dez. 2002)

[Hes 03] W. Hesse: Dinosaur Meets Archaeopteryx? or: Is there an Alternative for Rational's Unified Process? Software and Systems Modeling (SoSyM) Vol. 2. No. 4, pp. 240-247 (2003)

[H-S 02] C. W. Holsapple, K.D. Joshi: A Collaborative Approach to Ontology Design. CACM 45.2, pp. 42-47 (Feb. 2002)

[JBR 99] I. Jacobson, G. Booch, J. Rumbaugh: The Unified Software Development Process. Addison-Wesley 1999

[Jan 01] P. Janich: Wozu Ontologie für Informatiker? Objektbezug durch Sprachkritik. In: K. Bauknecht et al. (eds.): Informatik 2001 - Tagungsband der GI/OCG-Jahrestagung, Bd. II, pp. 765-769. books_372ocg.at; Bd. 157, Österr. Computer-Gesellschaft 2001

[M-K 02] H.C. Mayr , Ch. Kop: A User Centered Approach to Requirements Modeling. In: M. Glinz, G. Müller-Luschnat (Hrsg.): Modellierung 2002 - Modellierung in der Praxis - Modellierung für die Praxis, pp. 75-86, Springer LNI P-12 (2003)

[SMJ 02] P. Spyns, R. Meersman, M. Jarrar: Data modelling versus Ontology engineering, SIGMOD Record 31 (4), Dec. 2002

[UML 01] Unified Modeling Language (UML) 1.5 Documentation. OMG document ad/99/06-09. Rational Software Corp., Santa Clara, CA 2001. http://www.rational.com/uml/resources/ documentation