Vorlesung "Softwaretechnik"

Wiederverwendung

Lutz Prechelt

Freie Universität Berlin, Institut für Informatik

- Arten der Wiederverwendung
  - Fokus, Ziel/Vorteil
- Gegenstände der WV:
  - Anforderungen (→ Muster)
  - Schablonen, Checklisten
  - Entwurfsideen (→ Muster)
  - Komponenten
  - Methoden, Prozesse (→ Muster)
  - Werkzeuge
- Risiken und Hindernisse
- Beispiele für Muster
  - Prinzipien
  - Analysemuster
  - Benutzbarkeitsmuster
  - Prozessmuster
  - Anti-Muster

Teil 2

Lernziele

- Das Universum von Wiederverwendung verstehen:
  - Ziele, Arten, Vor- und Nachteile, Abwägungen und Risiken
- Verstehen, warum Muster ein, äh,
  Muster von Wiederverwendung sind
- Prinzipien der Softwaretechnik als Muster verstehen
- Muster für Domänenanalyse (im Unterschied zu Entwurfsmustern) verstehen

Wo sind wir?: Taxonomie "Die Welt der Softwaretechnik"

Freie Universität Berlin

Welt der Problemstellungen:

- Produkt (Komplexitätsprob.)
  - Anforderungen (Problemraum)
  - Entwurf (Lösungsraum)
- Prozess (psycho-soziale P.)
  - Kognitive Beschränkungen
  - Mängel der Urteilskraft
  - Kommunikation, Koordination
  - Gruppendynamik
  - Verborgene Ziele
  - Fehler

Welt der Lösungsansätze:

- Technische Ansätze ("hart")
  - Abstraktion
  - **Wiederverwendung**
  - Automatisierung
- Methodische Ansätze ("weich")
  - Anforderungsermittlung
  - Entwurf
  - Qualitätssicherung
  - Projektmanagement

Wo sind wir?: Wiederverwendung

- Einsicht: Etwas bekanntes wiederzuverwenden kann Qualität und Produktivität stark erhöhen und Risiko senken

- Prinzipien:
  - **Normales Vorgehen**: Vermeide radikales Vorgehen
  - **Universalität**: Fast alles lässt sich im Prinzip wiederverwenden
    - z.B. Anforderungen, Anforderungsmuster, Architekturen, Teilentwürfe, Entwurfsmuster, Testfälle, Dokumentschablonen, Vorgehensbeschreibungen, Checklisten, Prozessmuster
  - **Abwägung:** Wäge sorgfältig den Gewinn an Produktivität und (hoffentlich) Qualität ab gegen den Verlust an Flexibilität und Kontrolle

Arten der Wiederverwendung (1): Fokus

Mit welcher Orientierung kann man wiederverwenden?

- Produktorientiert
  - Anforderungen
  - Architekturen
  - Teilentwürfe, Entwurfsideen
  - Konkrete Komponenten
  - Testfälle
  - (siehe auch viele Einträge der nachfolgenden Folie)
- Prozessorientiert
  - Beschreibungen von Aktivitäten und Rollen
  - Methoden
  - Werkzeuge, Automatisierung
  - Sonstige Infrastruktur
    - z.B. Dokumentvorlagen
  - Maße
  - Erfahrungen
    - Verhalten von Kunden
    - Verhalten von Entwicklern
    - Firmenkultur
    - Projektdynamik
    - Erfolg von Prozessen

Arten der Wiederverwendung (2): Gegenstand

Was kann man wiederverwenden?

- Problem
  - Funktionale Anforderung
    - bewährte domänenspezifische Anforderung
    - Problemrahmen
    - Analysemuster
  - Nichtfunktionale Anforderung
    - z.B. Kriterien für Erlernbarkeit, Verfügbarkeit, Antwortzeiten etc.
  - Dokumentschablone, Formular
    - z.B. für Projektplan, Use Case, API-Dokumentation, Mängelbericht, u.v.a.m.
  - Checkliste
    - z.B. für Anforderungsermittlg., Analyse, Entwurf, Kodierung, Durchsicht, Test, Dokumentation
- Lösung
  - Konkrete Lösung
    - Dienst
    - Binärcode (Bibliothek, Komponente, Anwendung)
    - Quellcode
    - Rahmenwerk
  - Konkrete Lösungsidee
    - Entwurfsmuster
    - Benutzbarkeitsmuster
    - Prozessmuster
  - Allgemeine Lösungsidee
    - Methode
    - Prinzip
- Andere Mischformen
  - auch Muster sind Mischformen
  - z.B. Werkzeug, Produktfamilie

Arten der Wiederverwendung (3): Ziel

Mit welchem Ziel kann man wiederverwenden?

- Aufwandsverminderung jetzt
  - indem man etwas nicht erst erfinden/entwickeln muss
- Qualitätsverbesserung
  - indem man etwas Ausgereiftes benutzt
- Risikoverminderung
  - indem man die Eigenschaften schon vorher kennt
- Standardisierung
  - indem man das Gleiche vielfach verwendet
  - führt zu den obigen Wirkungen
- Aufwandsverminderung später
  - indem man nur 0 oder 1 Lösung anstatt N Lösungen pflegen muss
- Fokussierung
  - indem man sich mehr auf seine Hauptkompetenz konzentriert
  - und Nebensachen Anderen überlässt
- Beschleunigung des Markteintritts
  - Zusatzwirkung der Aufwandsverminderung
- Flexibilisierung
  - indem man ggf. mit geringeren Kosten die Lösung wechseln kann

Risiken der Produkt-/Lösungs-Wiederverwendung

Freie Universität Berlin

"Produktrisiko":

- Qualität ungewiss
  - Eventuell ist ein Produkt nicht hochwertig genug
    - in einer relevanten Hinsicht
  - Dadurch ist die Risiko- und Aufwandsverminderung ungewiss
- Eignung ungewiss
  - Man versteht evtl. nicht von vornherein, ob das Produkt die Anforderungen wirklich erfüllen kann
- Flexibilität eingeschränkt
  - Man kann Qualitätsmängel oder Funktionslücken evtl. nicht selbst korrigieren

"Lieferantenrisiko":

- Zwangsstandardisierung
  - Bei fremd zugelieferten Produkten geht die künftige Entwicklung evtl. in eine Richtung, die mir ungelegen ist
- Entwicklungsstillstand
  - Evtl. stellt der Hersteller die Fortentwicklung komplett ein

Beide gelten vor allem für die WV ausführbarer Komponenten

- Etwas weniger bei OpenSource-Komponenten
- Wiederverwendung von *Ideen* aller Art hat viel geringeres Risiko
  - ➔ **Muster!**

Beispiel für Risiko/Nutzen-Abwägung

Abwägung:

- **A:** Komponente K von Hersteller H zukaufen
  - Vorher nach Funktionalität auswählen
  - und Qualität überprüfen
- Vorteile von A:
  - Implementierung gespart
  - Qualitätssicherung weitgehend gespart
- Nachteile: Produktrestrisiko, Lieferantenrisiko

- **B:** Nur Anforderungen und Schnittstellen von K abgucken
  - aber Implementierung selber bauen
- Vorteile von B:
  - Funktionalität passt sicher(?) zu meinen Anforderungen
  - Kann nötige Qualität erzwingen
  - Habe die Fortentwicklung selbst in der Hand
- Nachteile: Höherer Aufwand (jetzt und später), Projektrisiko

Die Quantifizierung ist oft schwierig!

Hindernisse für Wiederverwendung

Wiederverwendung existierender Elemente (Produkt, Prozess):

- "Not invented here"
  - Ingenieure mögen evtl. fremde Dinge nicht, die sie auch selbst bauen könnten
- Ignoranz, intellektuelle Faulheit
  - Man weiß zu wenig darüber, was es alles gibt
- Auswahlaufwand
  - Evtl. gibt es Dutzende unbrauchbarer und nur 1–2 brauchbare Kandidaten
  - Dann ist der Auswahlaufwand hoch ("Nadel im Heuhaufen")
  - Zusammen mit verbleibendem Produkt- und Lieferantenrisiko gibt das manchmal den Ausschlag für Eigenentwicklung

Produktion wiederverwendbarer Elemente (Produkt, Prozess):

- Zusatzaufwand
  - Ein wiederverwendbares Teil hat 2–10x so viel Entwicklungsaufwand wie ein nur für ein Projekt gestaltetes Teil
- Flexibilitätseinschränkung
  - Wenn es mehrere Benutzer eines Teils gibt, kann ich es nicht mehr nach Belieben an meine künftigen Anforderungen anpassen
    - sondern muss Rücksicht nehmen, abstimmen, informieren etc.
- Geeignete Organisationsform nötig
  - heute oft: "Inner Source"

Wiederverwendung in der Softwaretechnik

Freie Universität Berlin

- Wiederverwendung ist das Erfolgsmodell in der SWT
- Die Softwaretechnik ist überwiegend eine Lehre von der Wiederverwendung:
  - Architekturen
  - Komponenten
  - Methoden und Notationen

- Gemessen an produzierter Funktionalität pro Zeiteinheit sind Programmierer heute dramatisch viel produktiver als vor einigen Jahrzehnten
  - Dieser Unterschied ist praktisch komplett auf Wiederverwendung zurückzuführen:
    - Bibliotheken, Komponenten (z.B. RDBMS), Infrastrukturen
    - Betriebssysteme und Werkzeuge
    - Sprachen und Methoden

Ist die Produktivität wirklich angestiegen?

Freie Universität Berlin

- Es wird immer mal wieder behauptet, Programmierer seien seit 40 Jahren nicht produktiver geworden.
- Diese Behauptung ist völliger Unfug
  - Sie beruht meist auf dem Maß "Programmzeilen pro Personenmonat"
  - Aber eine Programmzeile ist heute viel mehr Funktionalität wert
- Tatsächlich kann die gleiche Funktion (auf Anwendungsebene) heute meist viel schneller realisiert werden
  - Ein Datenpunkt dafür stammt von Gerald Weinberg:
    - Quality Software Management 1, 18.3.1, Seite 290
  - Er schrieb 1956 ein Programm zur Simulation von hydraulischen Netzen (Wasserleitungs-Netzen) mit ~500 Stunden Aufwand
  - 1979 schrieb er das gleiche Programm erneut:
    - 2,5 Std. Aufwand
  - Das ist eine Produktivitätsverbesserung von 20.000% oder ca. 25% jährlich!

Wiederverwendung für normales Vorgehen

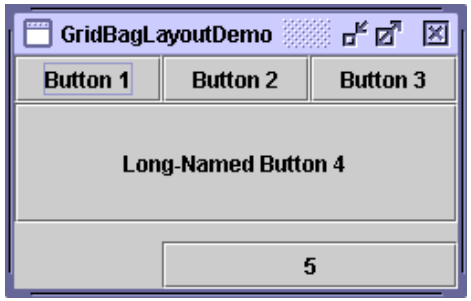
Freie Universität Berlin

Wie in "Die Welt der Softwaretechnik" besprochen,
sollte man bei SW-Entwicklung
nur dort radikal vorgehen, wo es wirklich nötig ist

- Wiederverwendung bedeutet meistens auch Abstützen auf Erfahrungen und fördert somit meistens (nicht immer!) das normale Vorgehen:
  - Aus Sicht des normalen Vorgehens ist an der Wiederverwendung nur die **Risikosenkung** von Interesse
  - Die Kostensenkung und die restlichen Vorteile gibt es quasi gratis dazu
- Es folgen Beispiele für diese Sichtweise:

Wiederverwendung von bewährten Komponenten als normales Vorgehen

- Eine komplette Softwarekomponente benutzen, *die man schon mehrmals zuvor in anderen Projekten für ähnliche Zwecke erfolgreich verwendet hat*
  - z.B. eine Klasse, ein Modul, ein Subsystem



- Vorteile / Normalität:
  - N1 Wir wissen aus Erfahrung, wie man die Komponente benutzt
    - und brauchen es nicht erst mühsam zu erlernen
  - N3 Wir kennen die typischen Probleme dabei (Risikofaktoren)
    - z.B. Defekte, Seltsamkeiten
  - N4 Wir verstehen, wofür die Komponente geeignet ist und wo ihre Grenzen liegen (Anwendbarkeitsbereich)
    - Funktionalität, Qualitätsmerkmale, Kapazitätsverhalten, Leistungsgrenzen
  - N5 Wir dürfen zuversichtlich sein, ein an dieser Stelle gut funktionierendes System zu erhalten

Risikosenkung

Wiederverwendung von bewährten Entwurfs-überlegungen/Entwürfen als norm. V.

- Eine Softwarekomponente nach den gleichen Überlegungen konstruieren, wie man es schon mehrmals zuvor in anderen Projekten für ähnliche Zwecke erfolgreich getan hat
  - z.B. Einsatz eines Architekturmusters oder Entwurfsmusters

- Vorteile / Normalität:
  - N1 Wir wissen, wie man die Überlegungen anwendet
  - N2 Wir verstehen, was das Wesen des Musters und seiner Nützlichkeit ausmacht (Erfolgsfaktoren)
  - N3 Wir kennen die typischen Stolperstellen beim Einsatz (Risikofaktoren)
  - N4 Wir verstehen, welche Ziele das Muster erreichen hilft und welche nicht oder wo es gar stört (Anwendbarkeitsbereich)
  - N5 Wir dürfen zuversichtlich sein, ein System mit den an dieser Stelle erwarteten Eigenschaften zu erhalten

Risikosenkung

Wiederverwendung von bewährten Prozess-elementen als normales Vorgehen

- Im Projekt ein Verfahren verwenden, das man bereits mehrmals zuvor in ähnlichen Zusammenhängen für ähnliche Zwecke erfolgreich verwendet hat
  - z.B. Durchsichten, Testautomatisierung, Objektmodellierung

- Vorteile / Normalität:
  - N1 Wir wissen aus Erfahrung, wie das Verfahren gemacht wird
    - und brauchen es nicht erst mühsam zu erlernen
  - N2 Wir verstehen, worauf es dabei ankommt (Erfolgsfaktoren)
    - weil wir aus früheren Fehlern gelernt haben
  - N3 Wir kennen die typischen Probleme dabei (Risikofaktoren)
    - weil wir aus früheren (Beinahe)Fehlschlägen gelernt haben
  - N4 Wir verstehen, wofür das Verfahren geeignet ist oder nicht und was es erreichen kann oder nicht (Anwendbarkeitsbereich)
    - z.B. übertreiben wir es nicht

Analog für Wiederverw. von z.B. Anforderungen oder Werkzeugen

Muster, Muster, Muster

- Vor ca. 1994 sprach man von Wiederverwendung meist nur auf der Ebene von Programmcode
  - Andere Ebenen, wie Anforderungen oder Entwurf galten als sehr schwierig
  - Die Wiederverwendung von Methoden schien nicht so bedeutsam
  - Wiederverwendung wurde oft als ein uneingelöstes Versprechen angesehen

- Dann erschien die Idee von Entwurfsmustern:
  - Paar aus Problem- und Lösungsbeschreibung
  - Abstrakter (und deshalb viel flexibler) als eine konkrete Lösung
- Diese Idee hat sich inzwischen in vielen Bereichen bewährt
  - Deshalb nun dazu ein paar weitere Beispiele

Arten von Mustern

- Anforderungen
  - Analysemuster 🡐
  - Akzeptanzkriterien
- Entwurf
  - Entwurfsmuster ✓
  - Architekturstile/-muster ✓
  - Referenzarchitekturen ✓
  - Produktfamilien
- Benutzungsschnittstellen
  - Benutzbarkeitsmuster 🡐
- Management, Vorgehen
  - Prozessmuster 🡐
  - Best practices
  - Standards
- Allgemein
  - **Prinzipien** 🡐
  - Notationen 🡐

- Ein paar Sorten (✓) haben wir schon besprochen
- Zu den markierten (🡐) folgen nun Erläuterungen/Beispiele

(Die Liste ist sehr unvollständig)

Prinzip, Methode, Verfahren: Alles Muster!

Freie Universität Berlin

- **Prinzip**: Ein Grundsatz, an dem man sein Handeln orientiert
  - Sehr abstrakt; sehr allgemein; allgemein gültig
  - **Bleibt lebenslang richtig und nützlich!**
- **Methode**: Planmäßig angewandte und begründete Vorgehensweise zum Erreichen festgelegter Ziele
  - Abstrakt; meist auf einen Bereich spezialisiert
  - **Bleibt einige Jahre (ca. 10–20) lang relevant**
- **Verfahren**: Präzise und recht konkrete Vorgehensvorschrift
  - **Für sehr spezifischen Bereich**

- Folgerung: **Lernen Sie virtuos mit Prinzipien umzugehen!**
  - Gleich schauen wir uns ein paar Prinzipen kurz als Muster an, die in der Softwaretechnik unentwegt wiederkehren: Abstraktion, Strukturierung, Hierarchisierung, Modularisierung, Lokalität, Konsistenz, Angemessenheit, Wiederverwendung

- Frage: Warum sind Methoden als Muster anzusehen?

Prinzip: Abstraktion

- Beschreibe X durch etwas einfacheres, das aber hinsichtlich der relevanten Eigenschaften gleich ist
  - Insbesondere: Beschreibe viele Exemplare durch einen Typ
- ! Abstraktion ist *die* Zentralidee der gesamten Informatik

- Beispiele:
  - Eine Prozedur in einem Programm ist eine Abstraktion einer Anweisungsfolge
  - Eine Temperaturangabe ist eine Abstraktion der Molekularbewegungen in einem Gegenstand

- (Achtung, Obiges ist eine stark verkürzte Darstellungsform):

Anmerkung: Beschreibungsform

- Um die vorherige Folie in ein richtiges Muster zu verwandeln, müsste man einiges ergänzen:
  - Anwendungsbereich
  - Klarere Trennung von Problem und Lösung
  - Vorteile
  - Nachteile und Abwägungen
  - Mögliche Variationen

- Ich gehe davon aus, dass es bei den Prinzipien reicht, sie überhaupt bewusst zu machen
  - und spare uns aus Zeitgründen die genauere Diskussion

Prinzip: Strukturierung

- Mache etwas Kompliziertes verständlich, indem Du den Einzelteilen ausdrücklich eine bestimmte Rolle oder Bedeutung im Ganzen zuweist

- ! Strukturierung verwendet immer auch Abstraktion

- Beispiele:
  - Erkläre den Softwareentwicklungsprozess als zusammengesetzt aus Anforderungsbestimmung, Entwurf, Implementierung, Test etc.
  - Erkläre eine Sprache durch eine Grammatik

Prinzip: Hierarchisierung

- Schaffe Übersicht bei einer großen Zahl von Teilen, indem Du
  - Jeweils einige Teile zusammenfasst
  - Und nötigenfalls jeweils Zusammenfassungen wieder als Teile betrachtest
  - In solch einer Weise, dass jede solche Zusammenfassung eine Bedeutung bekommt und zu einem Begriff wird
- Andere Sicht: Ordne eine Menge von Teilen als Blätter in eine Baumstruktur und mache dir die inneren Knoten zunutze
- ! Hierarchisierung ist ein Spezialfall von Strukturierung
- Beispiel:
  - Zusammenfassen von Dateien in Verzeichnisbäumen

Prinzip: Modularisierung

- Mache ein System mit einer großen Zahl von Funktionsteilen konstruierbar, indem Du
  - das gedachte Ganze zerlegst in nichttriviale Teile ("Module")
  - so dass jedes Modul einen klar beschriebenen Zweck erfüllt ("Schnittstelle", "Interface"),
  - das Verstehen und Benutzen eines Moduls von außen einfacher ist als das Verstehen aller seiner inneren Teile (Komplexitätsreduktion)
  - und ein Modul auf möglichst wenig Eigenschaften anderer Module angewiesen ist ("geringe Kopplung", "Trennung von Belangen").

- ! Modularisierung ist ein Spezialfall von Hierarchisierung und von Abstraktion

- Beispiel:
  - Batterie: Interface sind Bauform, Spannung, Kapazität; Innereien sind Bauart (z.B. Alkali, NiMH), Gehäusematerial, etc.

Prinzip: Lokalität

- Versammle alle Informationen, die zum Verstehen eines Teils oder einer Eigenschaft nötig sind, möglichst an einem Ort

- Beispiele:
  - Javadoc (Code und Dokumentation an einem Ort)
  - Java (versus C++: Klassendeklaration und Implementierung in einer Datei)

Prinzip: Konsistenz

- Handhabe gleichartige Dinge möglichst stets auf die gleiche Weise, um Verwirrung und Irrtümer zu vermeiden

- ! Konsistenz erfordert zuvor oft die Bildung einer Abstraktion
  - Gleichartigkeit

- Beispiele:
  - Methodennamen sind immer Imperative
  - Ein Dialog kann immer mit der Taste ESC abgebrochen werden
  - Warnungen gelb anzeigen, Fehlermeldungen rot

Prinzip: Angemessenheit

- Bei der Auswahl einer Lösung zu einem Problem ist zu berücksichtigen, inwieweit der erzielte Nutzen den betriebenen Aufwand rechtfertigt

- ! Angemessenheit ist ein Handlungsprinzip aller Ingenieure

- Beispiele:
  - Komplexe technische Apparatismen nur einsetzen, wo nötig
    - z.B. Simple Design bei XP
  - Gegenbeispiel ist unnötige Flexibilität:
    → schlechte Sicherheit oder Überforderung vieler Benutzer_innen
    - evtl. auch die komplexe Infrastruktur
      selbst einfacher Webanwendungen

Prinzip: Wiederverwendung

- Vermeide die Konstruktion komplexer Teile oder Ideen
  - Suche, ob es ein gleichwertiges Teil schon gibt
  - Oder ein ähnliches, zu dem Du Deine Anforderungen abwandeln kannst
  - Oder ein allgemeineres, das Du passend ausprägen kannst

- ! Wiederverwendung ist ein Spezialfall von Angemessenheit

- Auch ein Prinzip: Wiederverwendbarkeit
  - "Vermeide die Konstruktion hoch spezialisierter Teile"

Notationen

Auch Notationen (z.B. UML, Programmiersprache u.a.) können als Muster aufgefasst werden:

- **Problem:** Softwaretechnik ist auf die Zusammenarbeit mehrerer angewiesen; deshalb müssen Begriffe und Aussagen eindeutig wiederholbar festgehalten werden können
- **Lösungsidee:** Notation: Darstellung relevanter Konzepte durch festgelegte Menge von Symbolen mit definierter Syntax und Semantik
- **Abwägungen:**
  - Notationen sind nur Hilfsmittel zu einem Zweck; jede Notation kann manche Dinge besser ausdrücken als andere; deshalb ist wichtig, dass die jeweilige Notation gut zur Aufgabe passt
    - z.B. domänenspezifische Sprachen (DSLs), in der Praxis ein wichtiges Element von Model-Driven-Architecture (MDA)
  - Andererseits sind Definition und Erlernen von Notationen aufwändig; deshalb sind oft allgemeine Notationen zweckmäßig

Arten von Mustern

- Anforderungen
  - **Analysemuster** ←
  - Akzeptanzkriterien
- Entwurf
  - Referenzarchitekturen
  - Architekturstile/-muster, Entwurfsmuster
  - Produktfamilien
- Benutzungsschnittstellen
  - Benutzbarkeitsmuster ←
- Management, Vorgehen
  - Prozessmuster ←
  - Best practices
  - Standards

- Allgemein
  - Prinzipien ←
  - Notationen ←

- Zu den markierten (←) folgen nun Erläuterungen/Beispiele

Analysemuster

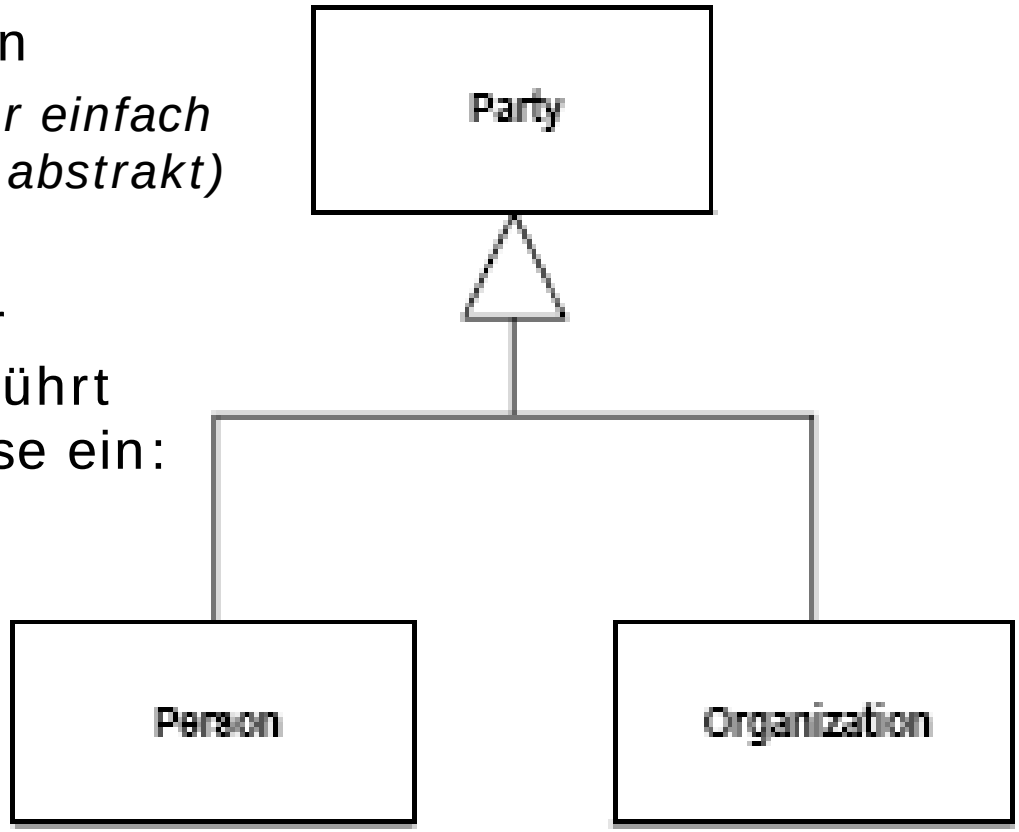
- Im Rahmen der Anforderungsanalyse tauchen bei der Objektbestimmung in vielen Domänen ähnliche Gruppen von Objekten mit ähnlichen Anforderungen auf

- Analysemuster identifizieren solche Gruppen von Klassen und beschreiben die Struktur ihres Zusammenwirkens
  - Der Übergang zu Entwurfsmustern ist fließend
  - Der Schwerpunkt liegt hier aber auf den Domänenobjekten und ihren Beziehungen
    - nicht auf ihren Schnittstellen
    - nicht auf Abläufen und Verhalten

- Wir betrachten hier nur ein einziges Beispiel:
  - Eine Gruppe von Mustern ("Mustersprache") zur Abbildung von Organisationshierarchien

Beobachtung 1: Personen versus Organisationen

- An vielen Stellen muss man in SW sowohl Personen als auch Organisationen behandeln
  - *(Achtung: Das fängt sehr einfach an, wird aber dann sehr abstrakt)*
- Meist sind weite Teile der Behandlung gleich, also führt man dafür eine Oberklasse ein:

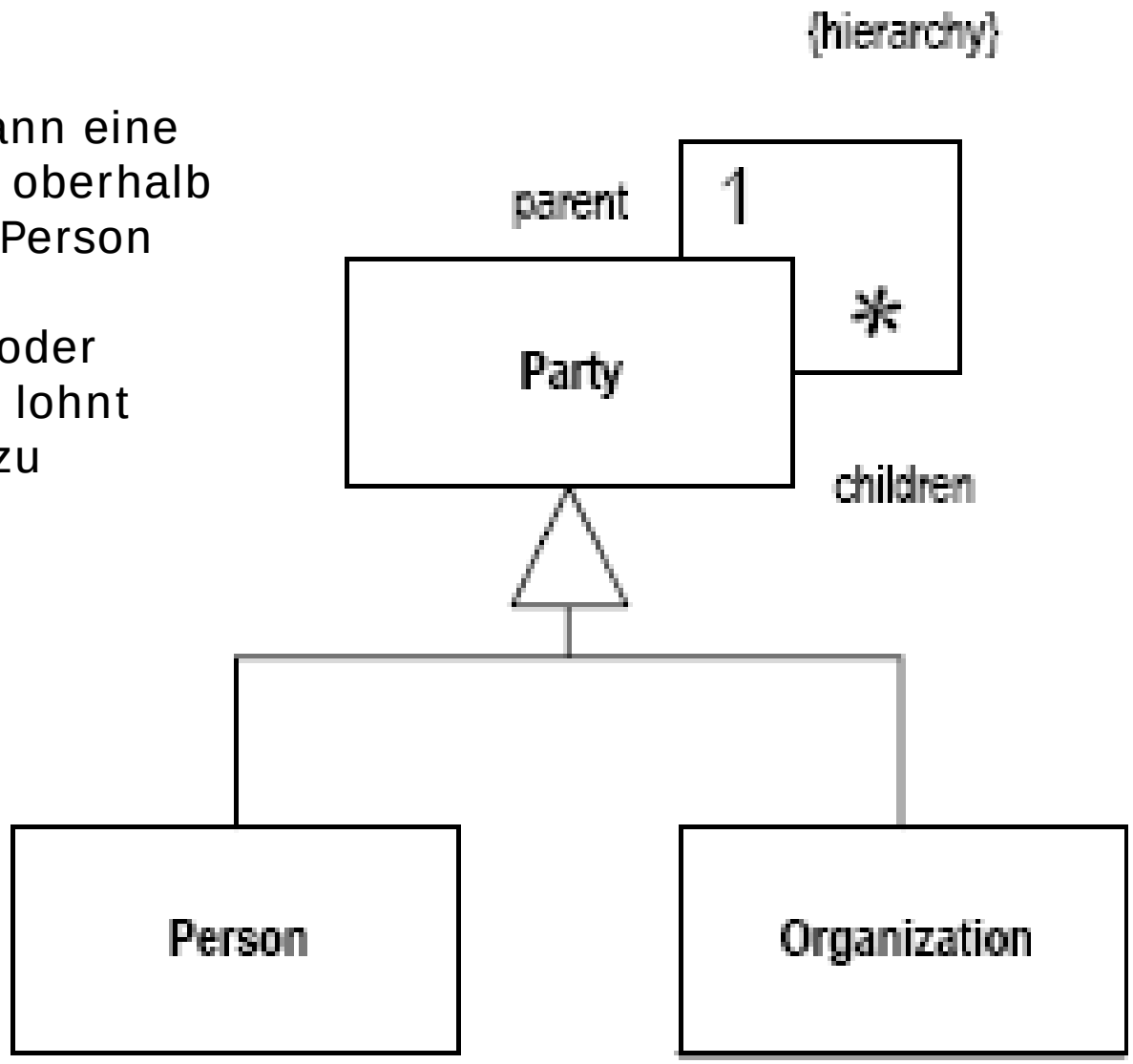


Beobachtung 2: Organisationen sind hierarchisch

- Die Organisationen haben meist eine Struktur aus Teilorganisationen
  - und die Teilorganisationen bestehen aus Personen

Beliebig tiefe Organisationshierarchie

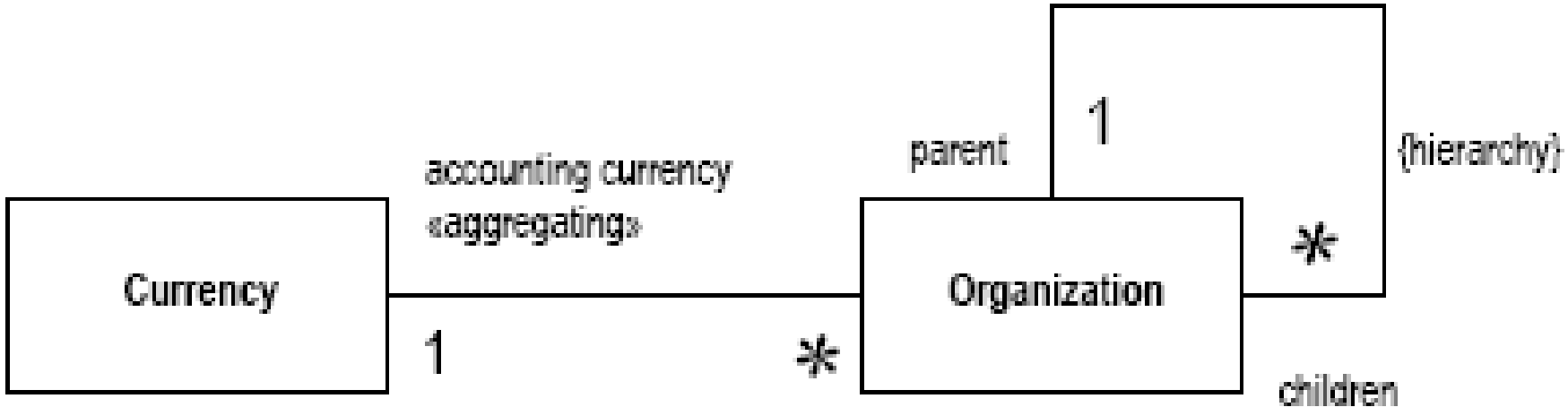
(Bei dieser Lösung kann eine Teilorganisation auch oberhalb der Blatt-Ebene eine Person sein.
Das kann erwünscht oder unerwünscht sein, es lohnt aber evtl. nicht, das zu verhindern.)



Soweit alles ähnlich wie bei Kompositum. Aber jetzt geht's noch weiter:

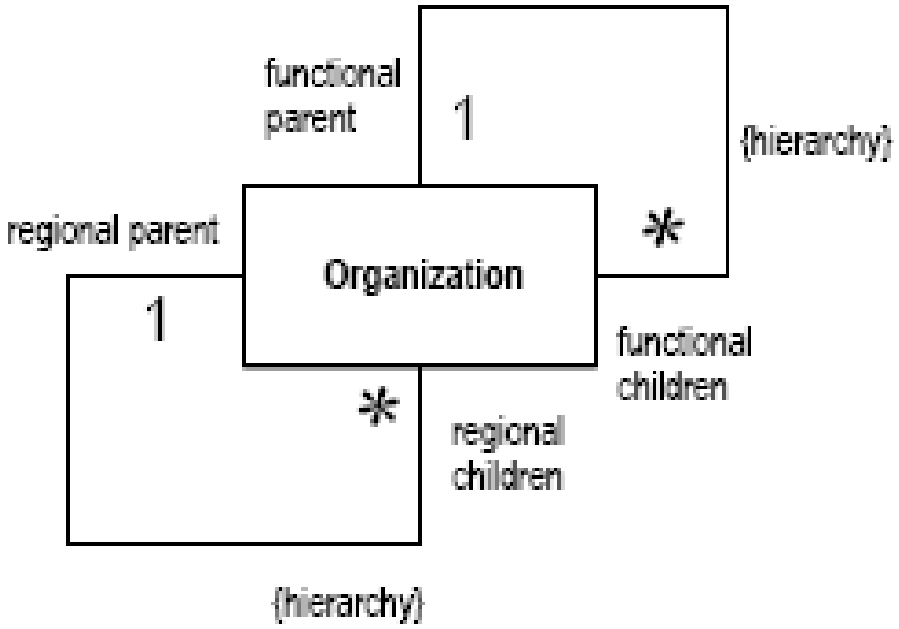
Propagation von Attributen

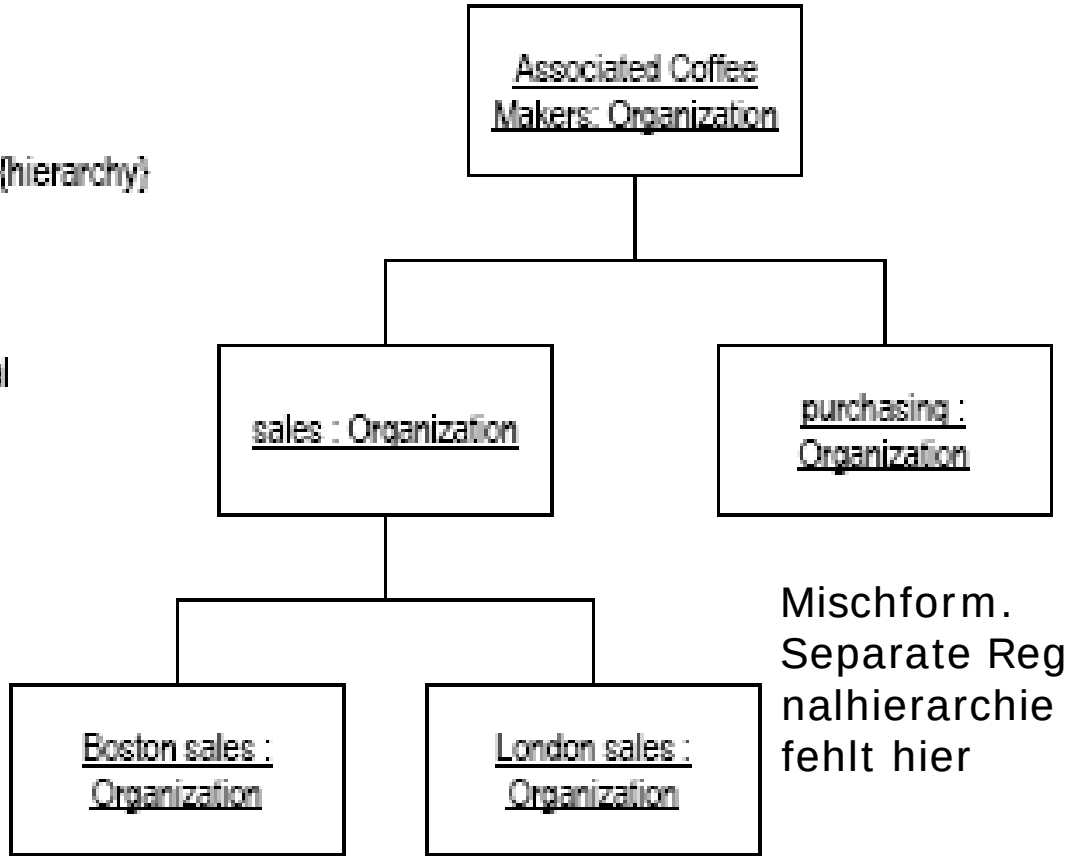
- Man kann Attributwerte entlang der Hierarchie nach unten weitergeben
  - Nicht direkt in UML ausdrückbar, deshalb als Stereotyp notiert
  - Es wird also nicht nur eine Exemplarvariable vererbt, sondern derer aktuelle Wert wird dynamisch von oben nach unten durchgereicht



Mehrfache Hierarchien

- Eventuell will man Teilorganisationen nach mehr als einem Kriterium in Hierarchien einordnen
  - z.B. Funktionshierarchie und Regionalhierarchie

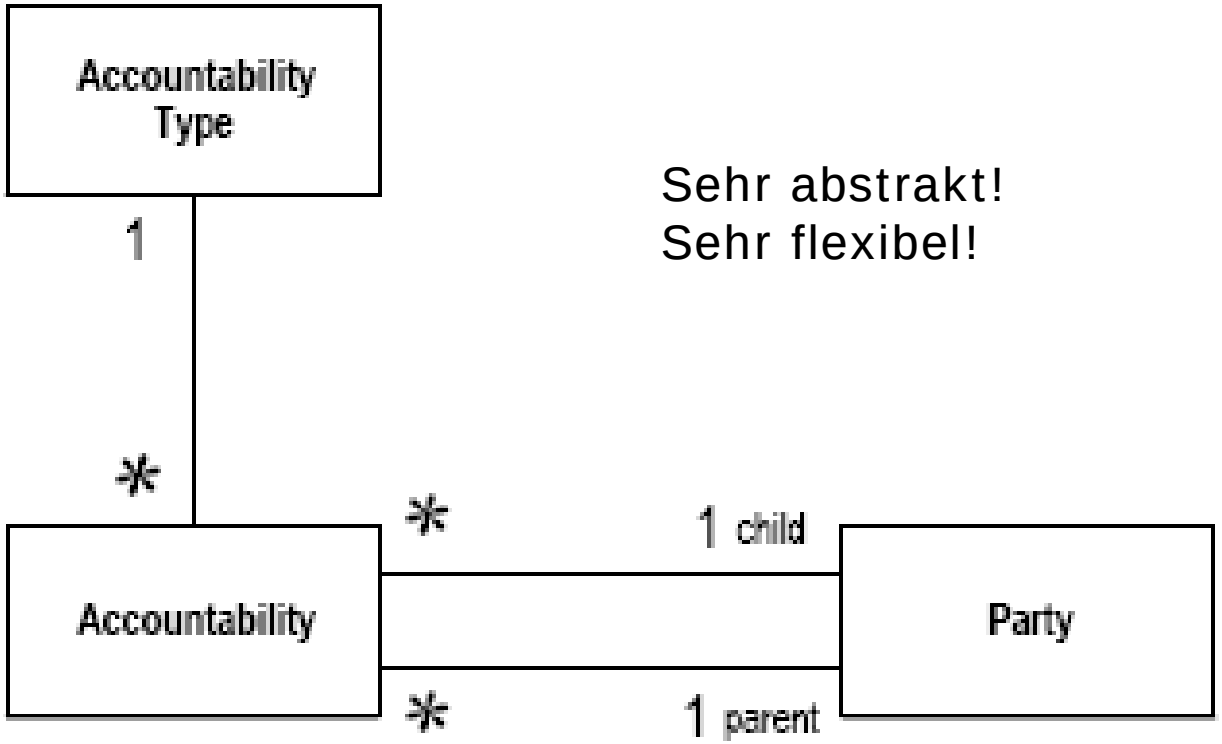




Mischform. Separate Regionalhierarchie fehlt hier

Verallgemeinerte Hierarchien: Verantwortlichkeit

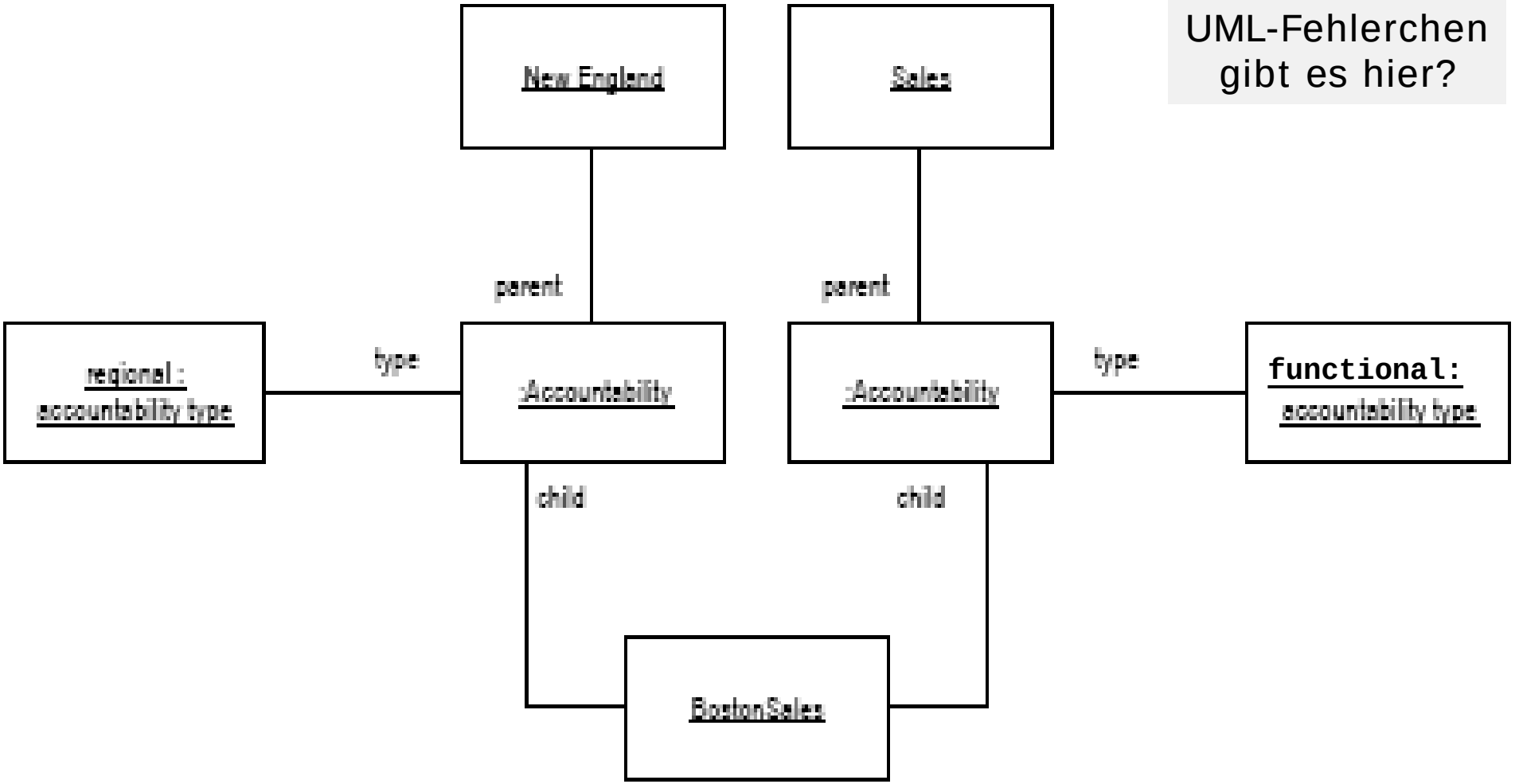
- Ein Verantwortlichkeitsobjekt (Accountability) verbindet eine Oberorganisation (parent) mit einer Unterorganisation (child)
  - Die Art der Verantwortlichkeit wird beschrieben durch ein Beziehungsobjekt (AccountabilityType-Objekt, z.B. "Funktionsgliederung", "Regionalgliederung")



Sehr abstrakt!
Sehr flexibel!

Verantwortlichkeit: Beispiel

Welche kleinen UML-Fehlerchen gibt es hier?

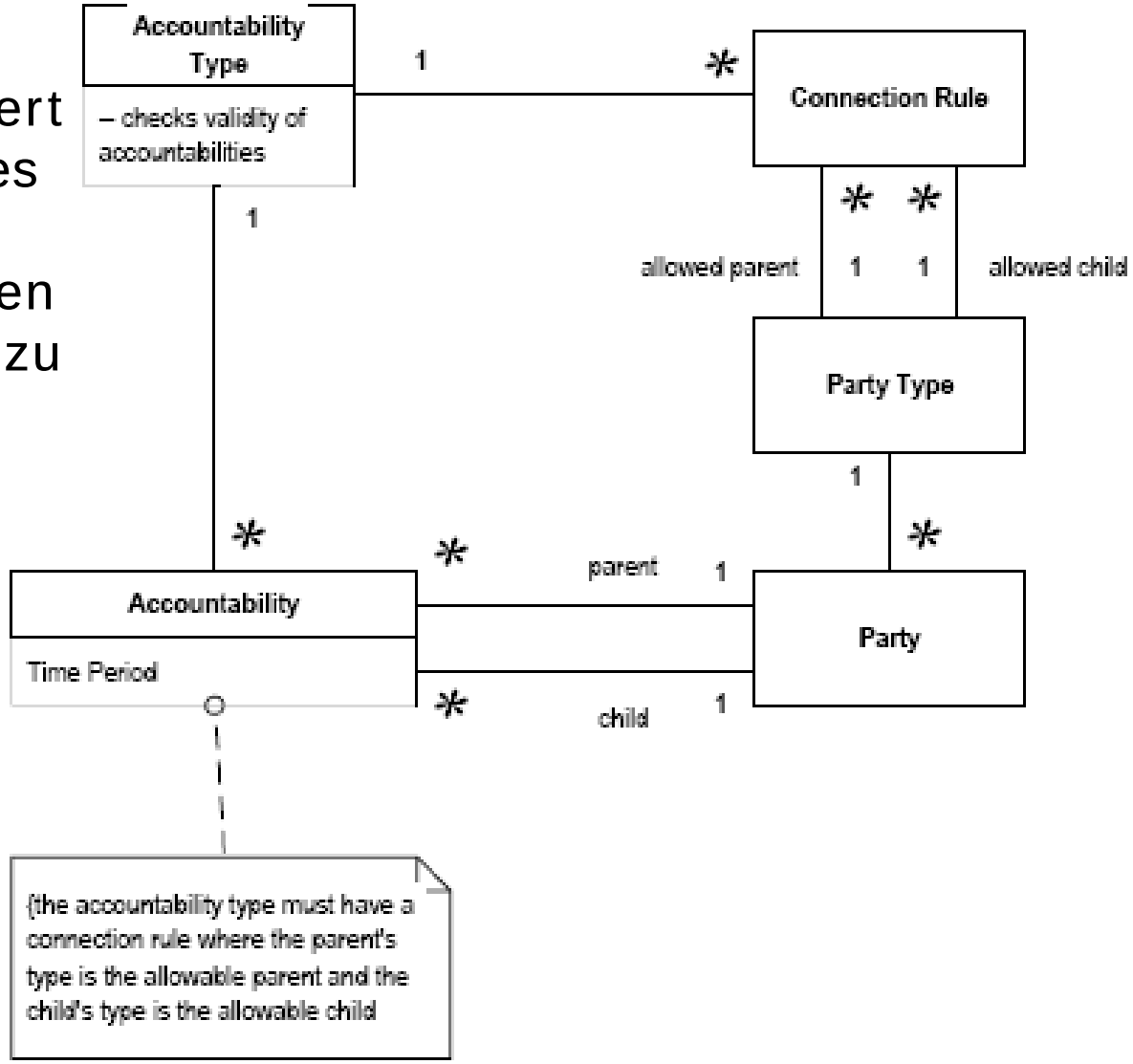


3 Anwenden

Hier geht's los

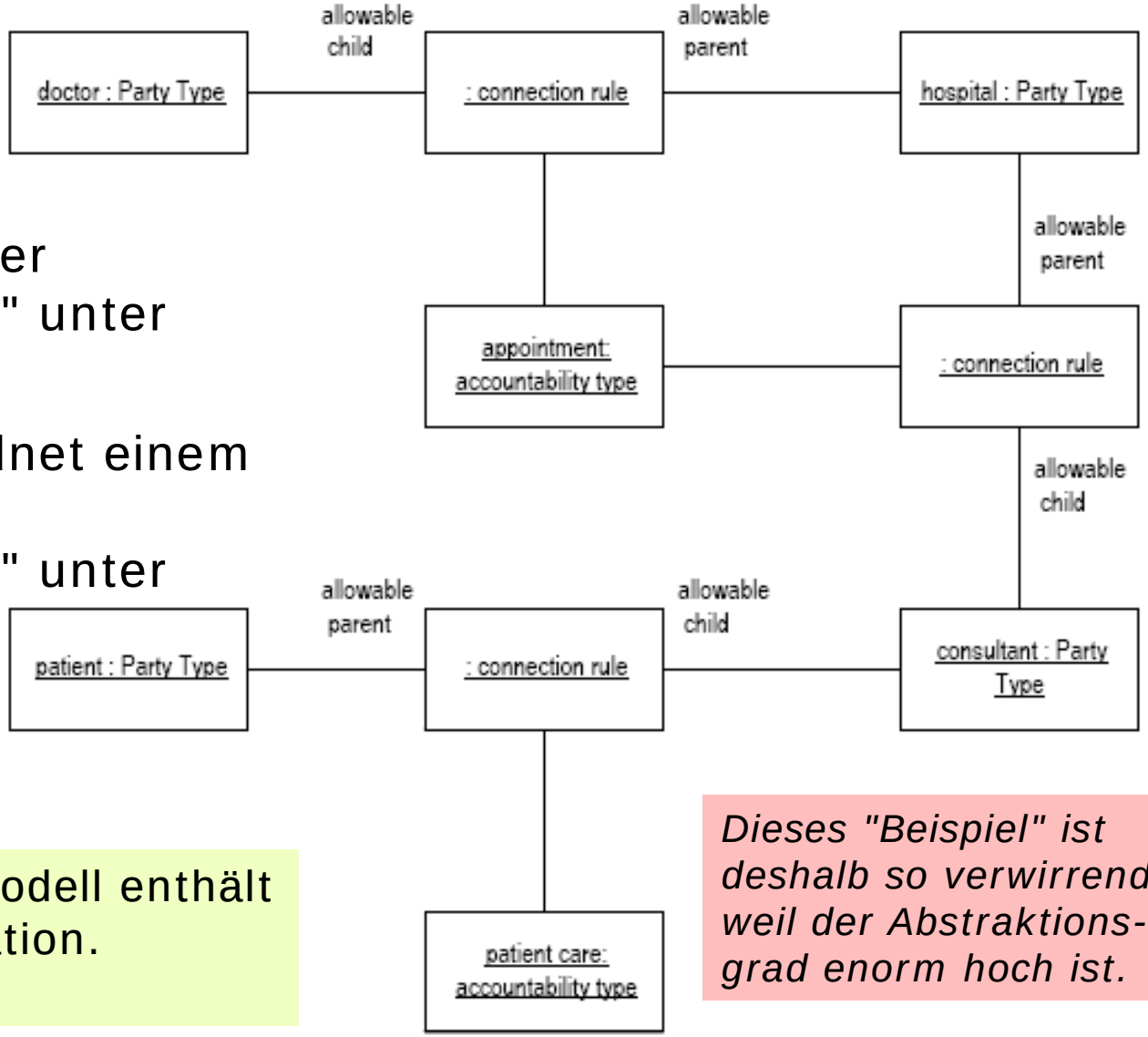
Verantwortlichkeiten mit Regeln

- Sollen Beziehungen zur Laufzeit verändert werden, empfiehlt es sich, die *Regeln* für erlaubte Beziehungen auch mit im Modell zu repräsentieren
  - ConnectionRule
  - PartyType



Verantwortlichkeiten mit Regeln: Beispiel

- "appointment" ordnet einem "hospital" einen "doctor" oder einen "consultant" unter
- "patient care" ordnet einem "patient" einen "consultant" unter
  - (das ist aber <u>keine</u> Organisations-hierarchie)

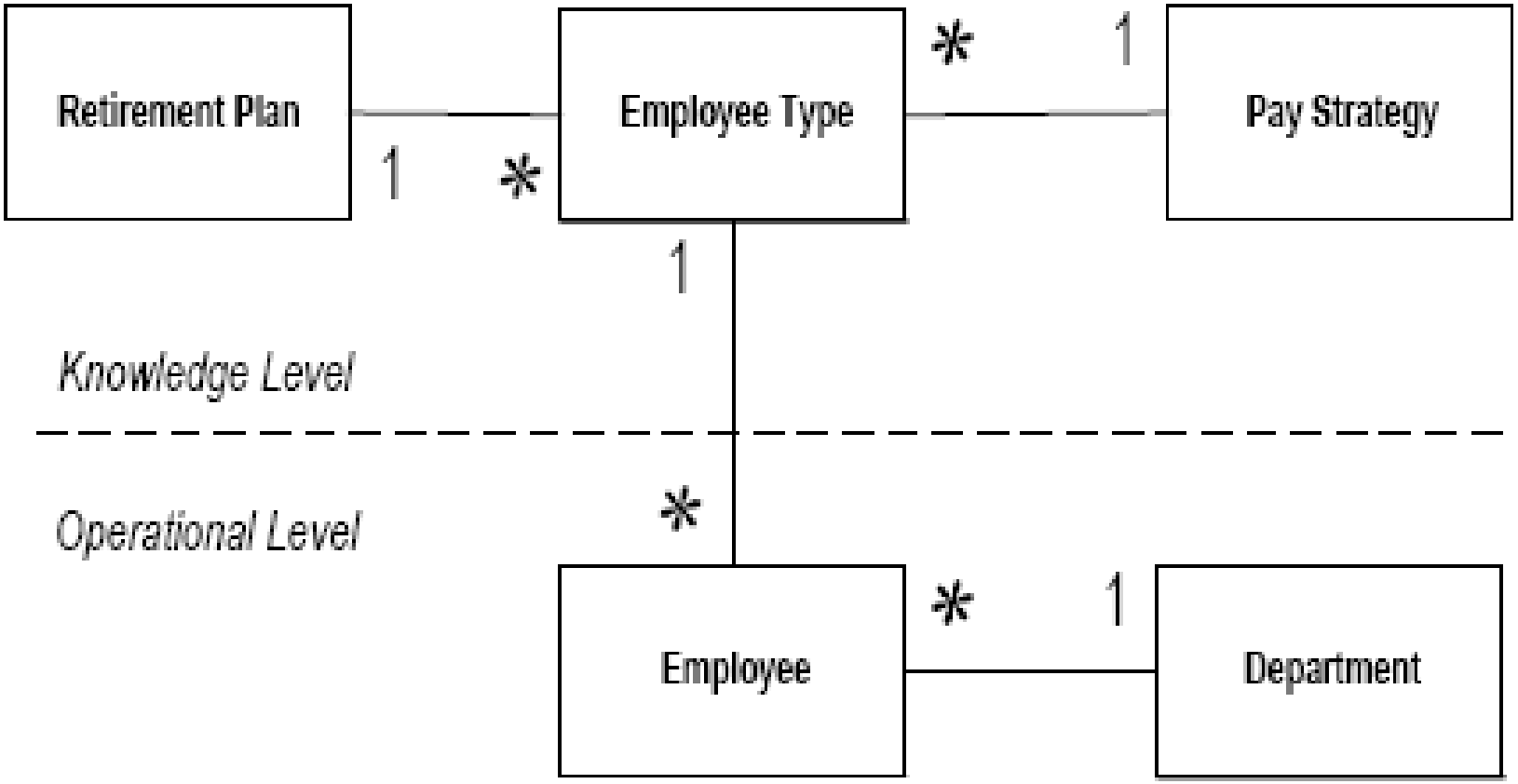


Achtung: Das ganze Modell enthält keine einzige Organisation. Alles nur Metamodell!

Dieses "Beispiel" ist deshalb so verwirrend, weil der Abstraktions-grad enorm hoch ist.

Verallgemeinerung: Metadaten

- Die Grundidee hinter ConnectionRule lässt sich oft anwenden: Trenne zwischen Inhaltsdaten (operational data) und Beschreibungsdaten (meta data, hier: "knowledge level")



Danke!

Weitere Beispiele folgen in Teil 2:
- Muster für Benutzbarkeit,
- Prozessmuster,
- Anti-Muster