Zerlegungsmethoden

Auch bei der Partitionierung lassen sich unterschiedliche Strategien ausmachen. Im Bereich der Software-Technologie sind dies die bekanntesten Methoden:

Slide 1

- funktionale Zerlegung
- datenorientierte Zerlegung
- objektorientierte Zerlegung

Funktionale Zerlegung

Funktionale (oder algorithmische) Zerlegung beginnt mit einer Problembeschreibung und teilt diese in durchzuführende Aufgaben, Prozesse oder Funktionen auf.

Slide 2

Die Unterfunktionen können wiederum aufgeteilt werden, bis die Aufgabe mit dem gewünschten Detaillierungsgrad beschrieben wurde.

Zu den funktionalen Zerlegungsmethoden gehören auch Datenflußdiagramme (obwohl der Name dies nicht impliziert!).

Datenorientierte Zerlegung

Bei der Partitionierung eines Systems entlang der Daten sind mehrere Ansätze möglich:

Slide 3

- rekursive Unterteilung des gesamten Datenbestandes in Mengen (prinzipiell der ER-Modellierung sehr ähnlich)
- rekursive Unterteilung der Daten in über- und untergeordnete Strukturen (vergleichbar mit hierarchischen Datenbanken)

Objektorientierte Zerlegung

Objektorientierte Zerlegung ist die umfassendste Methode (wobei *umfassend* sowohl positiv als auch negativ verstanden werden kann: einerseits ist sie leistungsfähig, andererseits auch komplex).

Slide 4

Das Grundprinzip besteht in einer Aufteilung des Systems (oder Problems) in Klassen und Objekte (Vorsicht: diese Begriffe werden nicht immer klar voneinander abgegrenzt benutzt!). Allerdings haben sich hier sehr viele Vorgehensweisen entwickelt.

Um die objektorientierte Zerlegung beurteilen zu können, soll hier zunächst der Begriff „objektorientiert“ vertieft werden.

Anmerkung: alle drei Zerlegungsmethoden können natürlich auch als *bottom-up*-Methoden oder als Mischung von *top-down*- und *bottom-up* verwendet werden!

Objektorientierte Systeme

Zentrales Bestandteil von objektorientierten Systemen ist das *Objekt*. Generell ist ein Objekt durch

Slide 5

- Attribute und
- Operationen

charakterisiert. Die konkrete Realisierung eines Objekts kann jedoch sehr unterschiedlich ausfallen, wobei die Hauptunterschiede zwischen den einzelnen zur Umsetzung benutzten Programmiersprachen in der Beschreibung der Operationen besteht.

Operationen

Grundsätzlich ergeben sich zwei Extrempunkte für die Beschreibung eines Objekts und insbesondere der Operationen des Objekts:

Slide 6

- jedes Objekt enthält eine eigene Beschreibung der Operationen
- die Operationen werden nicht im Objekt gehalten, sondern in einer getrennten Beschreibung, der *Klasse*

(Systeme und Programmiersprachen nach Punkt zwei sind weitaus häufiger anzutreffen. Grundsätzlich sind solche Systeme eher statischer Natur, während Systeme nach Punkt eins mehr Dynamik erlauben)

Attribute

Für Attribute gelten prinzipiell die gleichen Punkte wie für Operationen:

Slide 7

- Beschreibung der Attribute im Objekt enthalten
- Beschreibung der Attribute durch eine Klasse

Natürlich kann es sowohl bei Operationen als auch bei Attributen zwischen beiden Extrempunkten Mischformen geben, insbesondere die Forderung nach mehr Flexibilität resultiert in dynamischen Systemen nach Punkt eins.

Prototypbasierte Systeme

Systeme, in denen Objekte nicht durch eine statische Beschreibung ihrer Operationen (also ihres *Verhaltens*) charakterisiert sind, werden größtenteils als *prototypbasierte* Systeme umgesetzt:

Slide 8

- Objekte werden entweder komplett leer oder aus Prototypen erzeugt
- Attribute werden im Objekt selbst gehalten
- Operationen werden entweder im Objekt gehalten oder die Objekte verweisen auf ein *Prototypobjekt*, welches die Operationen realisiert

Charakteristika von prototypbasierten Systemem

Prototypbasierte Systeme haben eine Reihe von spezifischen Eigenschaften:

Slide 9

- das Verhalten eines Objekts kann durch Austausch des Verweises auf den Prototypen sehr einfach geändert werden
- auch die Eigenschaften der Prototypen lassen sich einfach ändern
- es gibt keine oder sehr geringe Möglichkeiten, das System statisch (d.h. beim Entwurf und nicht zur Laufzeit) zu analysieren

Klassenbasierte Systeme

Ein großes Anliegen gerade im Bereich sicherheitskritischer Systeme ist aber gerade die Möglichkeit zur kompletten statischen Analyse.

Slide 10

Hier eignen sich *klassenbasierte* Systeme besser, da eine Klassenbeschreibung statisch zur Entwurfszeit festgelegt ist und nicht geändert werden kann. Auch ist eine Änderung der Zuordnung *Klasse* $\Leftrightarrow$ *Objekt* nur in gewissen Grenzen möglich.

Der Schwerpunkt soll hier auf den klassenbasierten Systemen liegen, da sie mehr Möglichkeiten zur formalen Umsetzung von Designentscheidungen bieten (die Ausführungen orientieren sich dabei an der Programmiersprache *Eiffel*, die sehr viele objektorientierte Prinzipien umsetzt).

Der Klassenbegriff

Allgemein (und damit losgelöst vom Kontext der Software-Entwicklung) kann die Zugehörigkeit eines Objekts zu einer Klasse auf zwei Arten erfolgen:

Slide 11

- entweder werden alle Objekte aufgezählt, die zu einer Klasse gehören sollen, damit sind die Eigenschaften der Klasse durch die Objekte definiert,
- oder die Klasse wird als Definition der Eigenschaften benutzt, so daß nur Objekte der Klasse angehören, die diese Regeln erfüllen.

Der Klassenbegriff in der Software-Entwicklung

In der Software-Entwicklung kommen sicher beide Klassenbegriffe gleichberechtigt vor:

Slide 12

- in der Analysephase werden Objekte identifiziert und zu Klassen zusammengefaßt
- in der Designphase werden mittels der Klassen Regeln und Eigenschaften formuliert, nach denen sich Objekte dieser Klasse richten

Klassifizierung

Der Prozeß der Klassenbildung (also der Beschreibung der Eigenschaften der Klassen und der Abgrenzung einzelner Klassen) lehnt sich stark an Abstraktionsvorgänge an:

Slide 13

- es können konkretere Unterklassen von bestehenden Klassen gebildet werden (Vererbung)
- aus bestehenden Klassen können abstrakte Eigenschaften in eine Oberklasse verlagert werden

Vererbung

Vererbung als Bildung von Unterklassen hat zwei auf den ersten Blick widersprüchliche Bedeutungen:

Slide 14

Einschränkung: Die in der Oberklasse gemachten allgemeinen Aussagen werden in der Unterklasse konkretisiert und eingeschränkt

Erweiterung: Objekte der Unterklasse haben sowohl die Eigenschaften der Oberklasse als auch die Eigenschaften der Unterklasse

Vererbung

Damit hängen zwei weitere Sichten auf die Vererbungsbeziehungen zwischen Klassen zusammen:

Slide 15

- Vererbung des Interfaces
- Vererbung der Implementierung

Beide Sichtweisen (oder auch Nutzungsmöglichkeiten) sind auch strikt getrennt voneinander benutzbar, weswegen sie in manchen Programmiersprachen explizit unterschieden werden.

Klassenhierarchien und Typsysteme

Durch die Klassenbildung werden Hierarchien von Klassen aufgebaut, in denen bestimmte Regeln gelten:

Slide 16

- Gemäß den Abstraktionsprinzipien ist zugesichert, daß ein Objekt einer Unterklasse immer auch das Verhalten der Oberklassen aufweist.
- das Hinzufügen einer konkreten Unterklasse beeinflußt das Verhalten von Objekten der Oberklasse nicht

Klassenhierarchien und Typsysteme

Slide 17

- Zusicherungen, die in einer Oberklasse gemacht werden, können durch eine Unterklasse nicht gelockert werden
- Benötigte Vorbedingungen, die in einer Oberklasse definiert sind, können in einer Unterklasse nicht verstärkt werden

Man spricht auch von einem *Typsystem*, mittels dessen Hilfe Aussagen über das Verhalten von Objekten möglich ist, ohne aber die Implementierung zu betrachten.

Klassifizierung und Abstraktion

An dieser Stelle wird ein Punkt deutlich, an den objektorientierte und insbesondere klassenbasierte Systeme Vorteile gegenüber rein funktionalen oder datenorientierten Systemen haben:

Slide 18

Objektorientierte Systeme unterstützen *direkt* Abstraktion als Problemlösungsstrategie, indem sie Klassifizierungen von Objekten anhand von Eigenschaften zulassen.

Damit ist eine Bearbeitung des Problems in zwei Dimensionen möglich:

- Aufteilung in weniger komplexe Unterprobleme
- abstrakte Betrachtung und Beschreibung des Problems