Oberseminar Softwareentwicklung

„Data structures programs“ /
„Daten strukturieren Programme“

Pieter Hauffe

Data structures programs

„Die Wahl der Datenrepräsentation in Software beeinflusst die gesamte Entwicklung der Software .“

Gliederung

1. Was sind Daten?
2. Einfluss auf Programmstruktur
3. Einfluss auf Programmierkonzept/ -umgebung
4. Daten strukturieren
5. Allgemeine Prinzipien
6. Beispiele

1. Was sind Daten?

- (Informatik) Daten sind codierte Information
- Sinn jeglicher Programme ist es aus Daten andere Daten zu machen

► **Daten sind das zentrale Element in der EDV**

- Datenstrukturen sind sinnvolle Repräsentationen von Arbeitsdaten in Programmen

2. Programmstruktur

- **„Früher“:**
  - Eingabe, Verarbeitung und Ausgabe verstreut über den kompletten Quellcode
  - großer Aufwand für Programmierung von (nicht wieder nutzbaren) User-Interfaces, kaum davon getrennt Datenhaltung und –manipulation
  - Persistenz durch Ablage von Daten in Dateien

2. Programmstruktur

- **„Datenbanken“:**
  - zentralisierte Datenhaltung, auch über Anwendungsgrenzen hinweg
  - Datenmodellierung als eigene Disziplin
  - Query Language und Reporting als vollkommen neue Werkzeuge, auch in grafischer Ausprägung für Nicht-Programmierer benutzbar

2. Programmstruktur

- **„Objektorientierung“**
  - Zusammengehörende Daten und Programmlogik in Klassen zusammengefasst
  - Modularisierung des Codes, Wiederverwertbarkeit, Kapselung von Information, Vererbung, …
  - Persistenz durch Serialisierung oder DB

2. Programmstruktur

- Problem:
  - Unkenntnis oder Vorbehalte gegenüber „neuen“ Programmierparadigmen blockieren Weiterentwicklung
  - Vermittlungsprobleme
    - Einführung in Objektorientierung: Klasse „Gittarist“ mit Methode „gitarreSpielen()“ ?!?!
    - Relationale Datenbanken vs. Objektorientierung (Segmentierung, Methodenabbildung, …)

3. Programmierkonzept/ -umgebung

- Scripting
  - „kleine“ Aufgaben
  - Rapid Prototyping
  - Ad hoc Lösungen
- Funktionale Sprachen
  - Mathematische Probleme
- Logische Sprachen
  - Prädikatenlogische Probleme

3. Programmierkonzept/ -umgebung

- Visuelle Programmierumgebungen
  - sehr einfache Arbeit mit GUI
  - gute Unterstützung von OO
- Autorensysteme
  - Code wird verborgen
  - z.B. Multimediaanwendungen
- Anwendungsbezogene Sprachen
  - innerhalb eine gegebenen Anwendung

3. Programmierkonzept/ -umgebung

- Problem:
  - Programmierer versuchen mit ihrer Lieblingssprache alle Probleme zu erschlagen
  - Fehlender Einblick oder (unbegründete?) Vorbehalte gegenüber Programmierkonzepten resultiert in komplizierter oder unbenutzbarer Software, langer Entwicklung etc.

4. Daten strukturieren

- Zeitliche Steigerung der Anforderungen
  - Sinnvolle Variablennamen wählen
  - Funktionen zur Datenmanipulationen extrahieren
  - Strukturen definieren
  - Objekte identifizieren und Klassen entwerfen
  - Gebrauch von DB abwägen

5. Allgemeine Prinzipien

- Arrays benutzen (auch assoziative Arrays)
- Komplexe Strukturen kapseln
- „Fortgeschrittene Werkzeuge“ verwenden: Hypertext, XML, Domain specific languages, „Name=Wert“ Paare, Datenbanken, Design Patterns, etc.
- Vor dem Programmieren: Eingabe-, Ausgabe- und Zwischendaten verstehen und strukturieren

4. Beispiele

- Formbriefgenerierung:

  A) Klassisch:

```
print „Hallo “ + $vorname + „!“
print „Bitte überprüfen Sie Ihre Adressdaten:“
print $vorname + „ “ + $nachname
print $strasse + „ “ + $hausnummer
…
```

  Benutzerausgaben im Quellcode

  Problem: Änderungen erfordern Programmierer

4. Beispiele

- Formbriefgenerierung:

  B) externes Ausgabeschema + „Interpreter“:

```
Hallo $1 !
Bitte überprüfen Sie Ihre Adressdaten:
$2 $3
$4 $5
```

  Mit $i bezogen auf das i-te Element eines Benutzerdatensatzes (z.B. aus DB), wird durch den Interpreter mit Wert ersetzt

  Problem: fehlende Semantik der Zahlen, Änderung im DB Schema

  Besser: Verwendung von Schlüsseln, z.B. selbsterklärende DB Spaltennamen

4. Beispiele

- Finanzbuchhaltungssoftware:
  - Große Programme mit riesigem Aufwand für angemessene GUI, Bemühung von Spezialbibliotheken für Kalkulation, Datenbanken für Datenhaltung

    - vs -

  - Implementierung in Tabellenkalkulations-software, „natürliches“ Benutzerinterface, native Unterstützung von Finanzkalkulation, Datenspeicherung, etc.

4. Beispiele

- Umfrageprogramm:
  - Eingabe: n gleichförmige Datensätze
  - Ausgabe: ausgewählte Tabellen mit Summen, Durchschnitten, Prozenten etc.
  - Überlegung: interne Datenstruktur orientiert an Eingabe oder Ausgabe?

4. Beispiele

- Umfrageprogramm
  - Evolution:
    - 1.) Orientiert an Ausgabedaten, abgespeichert in einigen hundert Variablen: unüberschaubar, unwartbar und unglaublich fehleranfällig
    - 2.) Orientiert an Ausgabedaten unter Verwendung von Arrays: besser… aber unflexibel
    - 3.) Orientiert an Eingabedaten unter Verwendung einer Datenbank mit jedem Interview als einen Datensatz: Auswertung mit Anfragen

Quellen

- Jon Bentley „Programming Pearls“ (Second Edition), 1999 Addison-Wesley
- Wikipedia, http://de.wikipedia.org/ (Stand: April 2004)
- Steven S. Skiena „Data Structures and Programming“ (Lecture) http://www.cs.sunysb.edu/~skiena/214/lectures/lect1/lect1.html (Stand: April 2004)
- Jason E. Sweat „An Introduction to MVC Using PHP“ in „PHP|Architect“ (Vol 2, Issue 5 – Mai 2003) https://www.phparch.com/index.php
- Jim Williamson „Client Server Solutions“ (Lecture) Bolton Institute 2003