Imperative Sprachen

Strukturiertes Programmieren

gutes Software-Design (u.a.):
jedes Modul unabhängig von anderen
entwickelbar
testbar

→ Modul A mit so wenig Wissen über Modul B wie möglich geschrieben werden
→ Modul A veränderbar oder re-implementierbar, ohne Modul B verändern zu müssen

Imperative Sprachen

Strukturiertes Programmieren

PS-Hilfsmittel dazu:

1. Strukturierte Programmierung
2. Abstraktion
3. Module

Imperative Sprachen

1. Strukturierte Programmierung

Dijkstra (1969):“Structured Programming“:

Entwicklung eines Programms:

- *Die wichtigsten Funktionalitäten eines Programms aufstellen,*
- *sukzessive diese in kleinere Unterfunktionalitäten verfeinern,*
- *solange, bis Unterfunktionalität so klein, dass sie durch Basisoperationen einfach auszudrücken ist*

Subprobleme

1. *klein genug, sie zu verstehen*
2. *separat genug, sie unabhängig zu lösen*

Imperative Sprachen

Dijkstra‘s Beispiel damals:

1. *Stufe: (Problem als Ganzes)*

```
begin
        print first thousand prime numbers
end
```

2. *Stufe: (Aufteilung in 2 Subtasks, Datenstruktur soll Ergebnis der 1. Task an zweite liefern)*

```
begin
    variable table p
    fill table p with first thousand primes
    print table p
end
```

Imperative Sprachen

Dijkstra‘s Beispiel damals:

3. *Stufe: (Jede Subtask weiter verfeinern, unabhängig voneinander)*

```
begin
    integer array p[1:1000]
    make for k from 1 through thousand
        p[k] equal to the kth prime number      } Task 1
    print p[k] for k from 1 through1000         } Task 2
end
```

4. *Stufe: (Grundlegende Programmstruktur steht, jetzt Algorithmus für Task 1 finden, evtl. weiter verfeinern)*

Imperative Sprachen

Fundamentale Technik der Informatik:

teile und (be-)herrsche

Einflussfaktor 1: Vorgehensweise

1. Top-down-Programmierung
2. Bottom-up-Entwicklung
3. Prototyping
4. Interface und Spezifikation

Imperative Sprachen

1. Top-down-Programmierung:

   Sukzessive Zerlegung in Teilaufgaben, sukzessive Konkretisierung von anfangs abstrakt formulierten Lösungsideen

2. Bottom-up-Entwicklung:

   ausgehend von elementaren Operationen Modellierung von komplexeren Operationen durch Zusammensetzen bis gewünschte Funktionalität erreicht ist

3. Prototyping:

   einfache „Vorab“-Implementierung während Spezifikations- oder Entwurfsphase, zum Testen von Algorithmen,

   einfachere Detail-Absprache mit Auftraggeber

Imperative Sprachen

4. Interface und Spezifikation:

Interface: Beschreibung der Teile einer Programmkomponente, die für andere Komponenten sichtbar sind

Spezifikation: Beschreibung des Verhaltens einer Komponente, welches durch ihr Interface erkennbar ist

→ bei Änderung der Implementierung ohne Änderung von Interface und Schnittstelle keinerlei Auswirkung auf andere Komponenten

Einfaches Beispiel: Funktion, Interface = Funktionskopf, Spezifikation: Beschreibung Beziehung

Parameter ↔ Ergebniswert

Sqrt:	function sqrt(x:real):real	Interface
	$x>0$, dann $sqrt(x)*sqrt(x) \approx x$	Spezifikation

Imperative Sprachen

Einflussfaktor 2: Zerlegungsart

1. Funktionsorientierte Zerlegung:
   Aufgabenorientiert, große Aufgaben werden in kleinere zerlegt, nach Wirth
2. Datenorientierte Zerlegung:
   Struktur des Programms spiegelt Struktur der zu verarbeitenden Daten wider, nach Jackson
3. Modulorientierte Zerlegung:
   Programm als Sammlung von Modulen, Zerlegung orientiert sich an Zusammengehörigkeit von Daten und Operationen, nach Parnas
4. Objektorientierte Zerlegung:
   Programm als Sammlung von miteinander kommunizierenden Objekten, nach Meyer, Wirfs-Brock, Coad, Booch, Rumbaugh, Jacobson

Imperative Sprachen

Beispiel: Bankanwendung

„Wir verwalten das Geld unserer Kunden, indem wir Kunden-Konten führen, auf die die Kunden ihr Geld einzahlen, Geld abheben oder Überweisungen von diesem Konto tätigen. Die Kunden können sich natürlich jederzeit ihren Kontostand ansehen…..

Also, ein Kunde muss erst einmal ein Konto bei uns anlegen. Dazu benötigen wir seinen Namen und seine Adresse. Er bekommt dann die Kontonummer seines neuen Kontos mitgeteilt und muss diese bei Einzahlungen, Auszahlungen oder Überweisungen angeben….

Zum Überweisen? Dazu muss der Kunde das Zielkonto angeben, natürlich seine eigene Kontonummer und den Betrag, den er überweisen will. Ist noch genügend Geld auf seinem Konto, dann wird der zu überweisende Betrag bei ihm abgebucht und dem Zielkonto gutgeschrieben. Die Kontostände werden dann entsprechend angepasst. Das sieht der Kunde dann auf seinem nächsten Kontoauszug.

Imperative Sprachen

Beispiel: Bankanwendung

Funktionsorientierte Zerlegung:

- Bank
  - Konten verwalten
    - anlegen
    - löschen
    - einzahlen
    - abheben
    - anzeigen
    - überweisen
      - Betrag abfragen
      - Zielkonto abfragen
      - wenn Betrag < Bestand,
        Betrag von Konto
        Betrag auf Zielkonto

Imperative Sprachen

Beispiel: Bankanwendung

Datenorientierte Zerlegung (zum Beispiel):

- Bank
  - Konto
    - Eigentümer
      - Name (einlesen, ändern, abfragen)
      - Adresse (einlesen, ändern, abfragen)
    - Kontostand (abfragen, erhöhen, verringern)
    - Kontoauszug (drucken)
      - Kontenbewegung
        - Aufbuchung
        - Abbuchung

Imperative Sprachen

Beispiel: Bankanwendung - objektorientierte Zerlegung

„Wir verwalten das Geld unserer Kunden, indem wir Kunden-Konten führen, auf die die Kunden ihr Geld einzahlen, Geld abheben oder Überweisungen von diesem Konto tätigen. Die Kunden können sich natürlich jederzeit ihren Kontostand ansehen…..

Also, ein Kunde muss erst einmal ein Konto bei uns anlegen. Dazu benötigen wir seinen Namen und seine Adresse. Er bekommt dann die Kontonummer seines neuen Kontos mitgeteilt und muss diese bei Einzahlungen, Auszahlungen oder Überweisungen angeben….

Zum Überweisen? Dazu muss der Kunde das Zielkonto angeben, natürlich seine eigene Kontonummer und den Betrag, den er überweisen will. Ist noch genügend Geld auf seinem Konto, dann wird der zu überweisende Betrag bei ihm abgebucht und dem Zielkonto gutgeschrieben. Die Kontostände werden dann entsprechend angepasst. Das sieht der Kunde dann auf seinem nächsten Kontoauszug.

Imperative Sprachen

Objektorientierte Zerlegung:

1. Alle Substantive Klassenkandidaten
2. Alle Verben Methodenkandidaten
3. Klassenkandidaten: verschiedene Begriffe für ein- und dasselbe? nur einen behalten
4. Klassenkandidaten: substantiviertes Verb? dann evtl. Methodenkandidat
5. Klassenkandidaten: nur konkrete Ausprägung eines anderen Klassenkandidaten? Dann Objekt, nicht Klasse
6. Klassenkandidat: hat er Eigenschaften oder ist er nur Eigenschaft eines anderen Kandidaten? dann Attribut
7. Vater-Sohn-/Generalisierungs-Spezialisierungsbeziehungen suchen; evtl. neue Generalisierungsklasse einfügen
8. Sonstige Beziehungen suchen (Assoziationen)
9. Methodenkandidaten: Klassen zuordnen