Beispiel für systematische Programmentwicklung

- **Entwickeln**
- **Verbessern**
- **Effizienzbetrachtung**

Lösungsentwicklung

- **Bisher**
  - Problemformulierung (als Übungsaufgabe)
  - Lösung wird direkt Programm erstellt
    - **Hilfsmittel**: schrittweise Verfeinerung
      Wahl geeigneter Bezeichner
      Verwendung von Kommentaren
- **Realität**
  - Problemformulierung, Festlegung der Bedieneroberfläche etc. ist **wichtiger Teil** der Softwareerstellung
  - Softwareerstellung, -veränderung geschieht **in „Phasen“**
  - **viele andere Dokumente** ausser Quelltext sind nötig
  - Programm wird immer aus **Bausteinen** zusammengesetzt
  - bei Softwareerstellung wirken **viele Personen** mit
- **Softwaretechnik liefert geeignete Methoden und Techniken.**

Welche Lösung?

- **Problem → zugehörige Software (Programm)**
  es gibt (unendlich) viele Lösungen
- **Welche Lösung ist geeignet?**
- **Welche Eigenschaften soll die Lösung haben?**
  - Diese müssen explizit formuliert werden (Anforderungen)
  - Eigenschaften sind **Qualitätseigenschaften**
  - Suchen insbesondere ***effiziente(ste) Lösung***
- **Programme sind im allgemeinen falsch**
  - Große Programmsysteme sind immer falsch
  - Welche Maßnahmen sind notwendig, um Korrektheit zu „erzielen“?

Entwickeln

Präzisierung der Aufgabe

- **Aufgabenformulierung:**

  In einer Datei befindet sich eine unbekannte Anzahl von Zahlen. Diese soll gelesen und nach ihrer Größe ausgedruckt werden.
- **Unklarheiten**
  - Format: Wie ausdrucken?
  - Mehrfachvorkommnisse: erlaubt, mehrfach aufgeführt?
  - Sortierung: aufsteigend, absteigend?
  - Sortierung in einem Feld: max. Anzahl unbekannt?
    (internes Sortieren/externers Sortieren)
  - Welcher Typ der einzelnen Elemente: INTEGER, REAL?
  - Feste Randbedingungen: Zahlen < größte in der Basismaschine darstellbare Zahl
- **Neue Formulierung:**

  In einer Datei befinden sich max. 100 nicht notwendigerweise verschiedene INTEGER-Zahlen. Diese Zahlen sollen gelesen und der Größe nach aufsteigend sortiert werden und entsprechend ihrer Vorkommenshäufigkeit einzelnen ausgegeben werden und zwar je 10 pro Zeile.

Entwickeln

Erster Schritt

- **Schrittweise Verfeinerung**

```
MODULE Main;

CONST Max        = 100;
TYPE  Index      = [1..MAX];
      Zahlenfeld = ARRAY Index OF INTEGER;
VAR   anzahl     : CARDINAL;
      feld       : Zahlenfeld;

BEGIN
    Einlesen;
    Sortieren;
    Ausgeben;
END.
```


Entwickeln

Verfeinerung

```
PROCEDURE Einlesen () =
VAR x : INTEGER;
BEGIN
  anzahl := 0;
  x := GetInt();

  WHILE NOT EndOfFile() DO
    x := GetInt();
    anzahl := anzahl + 1;
    feld[anzahl] := x;
  END;

END Einlesen;
```

```
PROCEDURE Ausgeben () =
BEGIN

  FOR nr := 1 TO anzahl DO
    SIO.PutInt(feld[nr]);
    IF nr MOD 10 = 0 THEN
      SIO.Nl();
    END;
  END;

END Ausgeben;
```

Entwickeln

Sortierverfahren

- **Hauptteil Sortieren**
  - einfache Verfahren: n2
  - gute Verfahren: n log n
  - Sortierstrategien
    - Einfügen
    - Auswählen
    - Vertauschen ← Bubblesort
    - Verschmelzen

→ Vorlesung Datenstrukturen und Algorithmen

Entwickeln

Bubblesort

```
PROCEDURE Bubblesort () =
VAR hilf : INTEGER;
BEGIN

  FOR k := 1 TO anzahl DO
    FOR i := 1 TO anzahl-1 DO
      IF feld[i] > feld[i+1] THEN
        Vertausche(feld, i,i+1);
      END;
    END;
  END;

END Bubblesort;
```

Verbessern

Verbesserungen - 1. Schritt

i	a					
Undef.	16	33	94	82	6	58
1	16	33	94	82	6	58
2	16	33	94	82	6	58
3	16	33	82	94	6	58
4	16	33	82	6	94	58
5	16	33	82	6	58	94

1. *Durchlauf größte Zahl am richtigen Platz*
2. *Durchlauf zweitgrößte Zahl am richtigen Platz*

...

- **1. Brauchen nur maximal Anzahl-1 Durchläufe**
  - letzte Zahl ist dann automatisch am richtigen Platz
  - letzter Durchlauf ist überflüssig
- **2. Brauchen beim k-ten Durchlauf nur bis Anzahl-k zu laufen**
  - die restlichen Zahlen sind bereits sortiert
- **3. Abbruchkriterium einführen:**

  Nicht mehr durchlaufen, wenn im letzten Durchgang nicht vertauscht worden ist.

Verbessern

Verbesserung

```
PROCEDURE Bubblesort ()=
VAR getauscht : BOOLAEN;
BEGIN

  FOR k  := 1 TO anzahl-1 DO          (*1*)
    getauscht := FALSE;
    FOR i := 1 TO anzahl-k DO         (*2*)
      IF feld[ i ] > feld[ i+1 ] THEN
        Vertausche(feld, i, i+1);
        getauscht := TRUE;
      END;
    END;
    If NOT getauscht THEN EXIT;       (*3*)
  END;

END Bubblesort;
```

Verbessern

Verbesserung - 2. Schritt

k	i	a					
		16	33	94	82	6	58
1	5	16	33	82	6	58	94
2	1	16	33	82	6	58	94
2	2	16	33	82	6	58	94
2	3	16	33	6	82	58	94
2	4	16	33	6	58	82	94
3	1	16	33	6	58	82	94
	2	16	6	33	58	82	94
	3	16	6	33	58	82	94

Überflüssig bis Anzahl-k zu laufen, wenn im letzten Durchlauf nicht bis dorthin vertauscht wurde.

Verbessern

Verbesserung 2. Schritt

```
PROCEDURE Bubblesort() =
VAR hilf : INTEGER;
    rechtesEnde, letztesI: Index;
BEGIN
  letztesI := anzahl;

  REPEAT
    rechtesEnde := letztesI;
    letztesI := 0;
    For i := 1 TO rechtesEnde -1 DO
      IF feld[i] > feld[i+1] THEN
        Vertausche(feld, i, i+1);
        letztesI := i;
      END;
    END;
  UNTIL letztesI := 0;

END Bubblesort;
```

Rechtes Ende pro Durchlauf nicht nur um 1 verkleinern

Effizienzbetrachtungen

- **best-case-Analyse**
  - geordnet
  - 1 Schleifendurchlauf
  - n-1 Vergleiche, 0 Vertauschungen
  - hier Bubblesort gut !
- **worst-case-Analyse**
  - Umgekehrt geordnet
  - n-1 Durchläufe
  - k-ter Durchlauf: innere Schleife: (n-k)-mal vertauscht

$$\sum_{k=1}^{n-1} (n-k) = \frac{(n-1)(n-2)}{2} = \frac{n^2}{2} - \frac{3n}{2} - 1$$

Anzahl der Vergleiche/ Vertauschungen

- **Bubblesort**
  - $O(n^2)$ d.h. Konstante a, b, c mit
  - f^{wc}_{Zeit} (Bubblesort, FeldDerLaenge) $\leq an^2 + bn + c$