DeinProgramm — Programmieren für alle

Michael Sperber
sperber@deinprogramm.de

Der BWInf und ich

- 1986 Bundessieger 4. BWInf
- 1991 *Die Grafik-Connection*, mit 10 Endrundenteilnehmern
- 1997–2000 Aufgabenausschuß

Prämissen

- möglichst viele Teilnehmer
- *Programmier*wettbewerb
- aus Unterrichtsstoff erschließbar

Der Programmierunterricht und ich

- „AP Computer Science“ (1987/1988)
- . . .
- „Programmieren für Geisteswissenschaftler“ (1999)
- „Informatik I“ (1999/2000, 2000/2001)
- „Programmieren für Geisteswissenschaftler“ (2001/2002)
- **PISA**-Studie (2002)
- div. Fortbildungen, z.B. VWA Stuttgart (2003–)

Programmieren

(Realität)

$$f(x) := x + 5$$

```
PROGRAM f(INPUT, OUTPUT);

FUNCTION f(x : INTEGER) : INTEGER;
BEGIN
   f := x + 5;
END;

VAR x :  INTEGER;

BEGIN
    ReadLn(x) ;
    WriteLn(f(x))
END.
```

Programmieren

(Realität)

$$f(x) := x + 5$$

```
PROGRAM f(INPUT, OUTPUT);

FUNCTION f(x : INTEGER) : INTEGER;
BEGIN
   f := x + 5;
END;

VAR x :  INTEGER;

BEGIN
    ReadLn(x) ;
    WriteLn(f(x))
END.
```

Programmieren und Mathematik

„Ist n Element der Liste l?“

$$g(n,l) := \begin{cases} false & \text{falls } l \text{ leer} \\ true & \text{falls } n = \mathrm{first}(l) \\ g(n, \mathrm{rest}(l)) & sonst \end{cases}$$

Realität

```
TYPE
   ListType = ^NodeType;
   NodeType = RECORD
                 First : String;
                 Rest  : ListType
              END;

FUNCTION IsMember (Name : String;  List : ListType)
: BOOLEAN;
BEGIN
   IF List = NIL
      THEN IsMember := FALSE
   ELSE IF Name = List^.First
      THEN IsMember := TRUE
   ELSE
      IsMember := IsMember(Name, List^.Rest)
END;
```

Realität

```
PROGRAM NameOnList(Input, Output) ;
VAR List : ListType;
    Name : STRING;

PROCEDURE GetList(Var LIST : ListType);
BEGIN
  ...
END;

FUNCTION IsMember ...

BEGIN
    ReadLn(Name);
    GetList(List);
    WriteLn(IsMember(Name, List))
END.
```

Programmieren

(Realität)

- orientiert an Maschine
- 10% Problemlösung, 90% Syntax/Maschine

Primäre Anforderungen

- wenig Zeit im Unterricht $\Longrightarrow$ *klein und einfach*
- möglichst viele Beispiele/h $\Longrightarrow$ *kurze Programme*
- problemlösungsorientiert $\Longrightarrow$ *mächtig*
- motivierend $\Longrightarrow$ *attraktive Beispiele*

Syntax in Scheme

(*Operation* $\mathit{Operand}_1$ ... $\mathit{Operand}_n$)

$4 + 5 \Longrightarrow$ `(+ 4 5)`

$(4 + 5) \cdot 6 \Longrightarrow$ `(* (+ 4 5) 6)`

Definitionen in Scheme

(define (*Funktionsname Parameter*$_1$ ... *Parameter*$_n$)
  rechte Seite)

$$f(x) := x + 5$$

```
(define (f x)
  (+ x 5))
```

Scheme und Mathematik

$$g(n,l) := \begin{cases} \mathit{false} & \text{falls } l \text{ leer} \\ \mathit{true} & \text{falls } n = \mathrm{first}(l) \\ g(n, \mathrm{rest}(l)) & \mathit{sonst} \end{cases}$$

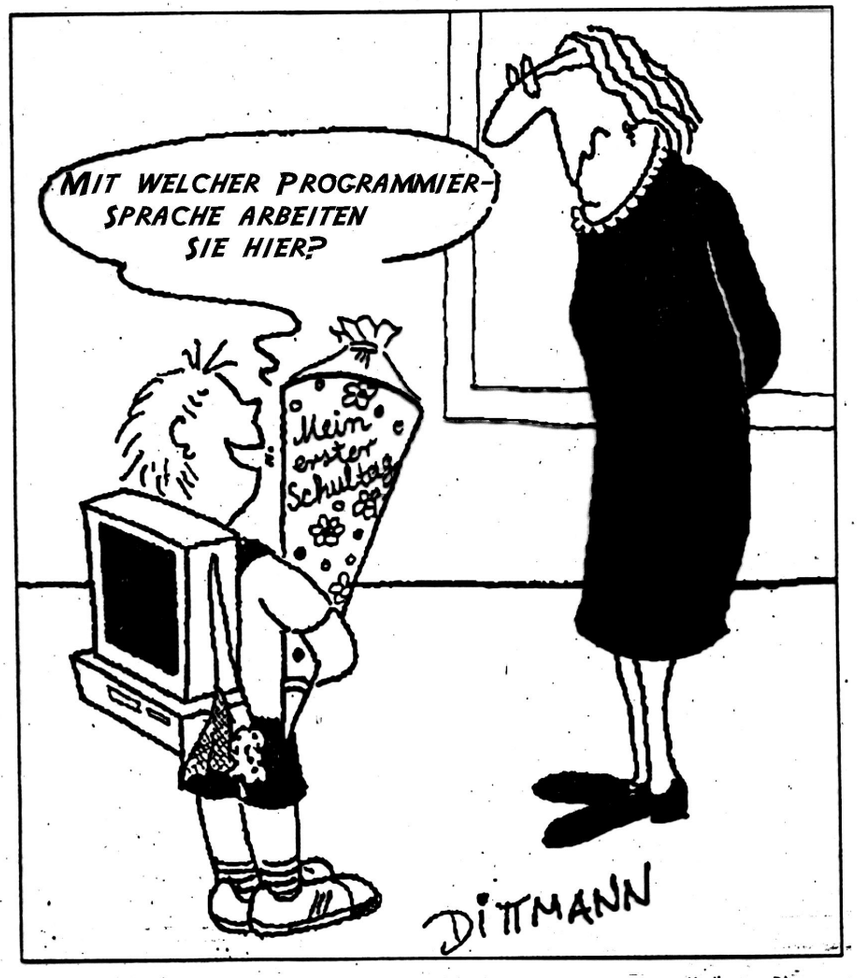
```
(define (guest name list)
  (cond
    ((empty? list)                    false)
    ((equal? name (first list))  true)
    (else (guest name (rest list)))))
```

10% Syntax, 90% Problemlösen

DrScheme und Teachpacks

- Grafik
- grafische Benutzerschnittstellen
- interaktive Web-Server
- geplant: Messen/Steuern/Regeln, Robotik, ...

Heilsversprechen Programmiersprache



Karikatur: Dittmann

Effekte traditionaller Programmierausbildung

- didaktischer Effekt ≈ 0
- viele Anfänger scheitern
- „Trial and Error“ hoffähig

Mathematik und Programmieren

PISA-Studie:

Mathematik-Kompetenz
→ *Mathematical Literacy*
→ Problemlösungskompetenz

Techniken:

- Modellierung
- Zerlegung zusammengesetzter Probleme

Programmierung:

- Modellierung
- „Abstraktion, Abstraktion, Abstraktion“
  (Grundtechnik zur Zerlegung zusammengesetzter Probleme)

Deutsche Schüler beim Problemlösen

PISA 2000:

(Deutsche Schüler) beim Programmieren

Programmkonstruktion

Eingabe $\Longrightarrow$ **Problem** $\Longrightarrow$ Ergebnis

Konstruktionsanleitungen

1. Problemanalyse
   - Verständnis
   - Vertrag
   - Beispiele/Testfälle
2. Funktionsdefinition
   - Gerüst
   - Schablone
   - Rumpf
3. Korrektheitsprüfung
   - Korrekturlesen
   - Syntaxüberprüfung
   - Test

Datengesteuerte Programmkonstruktion

„Die Daten bestimmen das Programm.“

zusammengesetzt	$\Rightarrow$	Selektion
gemischt	$\Rightarrow$	Fallunterscheidung
selbstbezüglich	$\Rightarrow$	rekursiver Aufruf
	...	

$\Longrightarrow$ *Wiederverwendbare Programmschablonen*

$\Longrightarrow$ *Rekursion Nebenprodukt des Datenmodells*

Nächstes Problem

```
...
y + 1 := x;
...
```

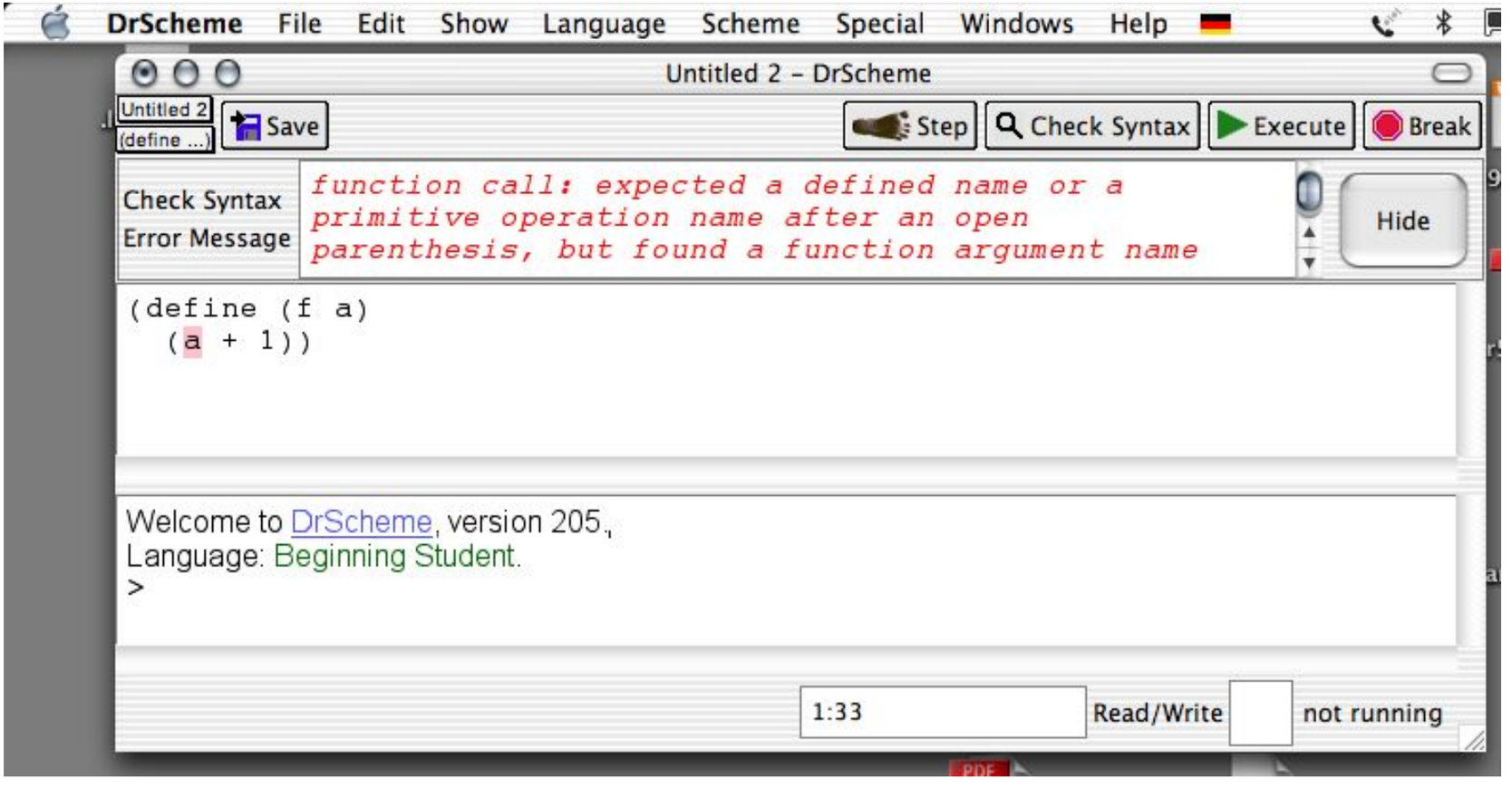
```
warning: expression used as a statement - value is ignored
parse error before ‘+’
```

Ursache:

Entwicklungsumgebungen für Profis gedacht, nicht Anfänger

DrScheme

- mehrere Sprachebenen
- präzise Fehlermeldungen
- interaktive Auswertung
- Break-Knopf



Erkenntnisse

- Programmierausbildung ist möglich
- sicher ab 5. Klasse
- erfordert grundlegendes Umdenken:
  1. Programmiersprache
  2. Programmierumgebung
  3. Didaktik

TeachScheme!

- Matthias Felleisen, Northeastern University
- Brown University, University of Chicago, University of Utah, Adelphi University ...
- Informatik-Unterricht an High Schools
- Programmierumgebung DrScheme
- *How to Design Programs*
- viele Universitäten
- $>$ 250 ausgebildete Lehrer
- in Arbeit: Transition zu Java

Programmieren als Grundkompetenz im Schulunterricht

- Programmierunterricht ab 5. Klasse
- Ausgangspunkt: Mathematik-Unterricht
- Anwendungen in anderen Fächern
- *nicht*: Ersatz für Informatik-Unterricht

- Lehrmaterialien
- Weiterentwicklung von DrScheme
- Software für Unterrichtsbeispiele
- Lehrer-Aus- und -Fortbildung

Projektpartner

- Erwin Bauer
  (freiberuflicher Systemanalytiker, Berater und Trainer)
- Prof. Herbert Klaeren, Uni Tübingen
- Prof. Helmut Schwichtenberg, LMU München
- TeachScheme!
- Lehrer in Baden-Württemberg, Bayern, Rheinland-Pfalz

Zusammenfassung

- Programmierausbildung für alle ist möglich
- . . . erfordert aber Umdenken
- . . . Werkzeuge
- . . . und systematische Didaktik

`http://www.deinprogramm.de/`