# Programmierung in der Lehrerausbildung

Bettina Timmermann, TU Dresden

**Abstract:** In diesem Artikel wird der Frage nachgegangen, welche Rolle die Programmierung und die Vermittlung von Programmiersprachen in der Lehrerausbildung spielen soll. Im Vordergrund stehen dabei grundlegende Konzepte der Problemlösung und deren Umsetzung in verschiedenen Programmierparadigmen. Außerdem wird über Erfahrungen berichtet, die an der TU Dresden in der seit Wintersemester 1991/1992 durchgeführten Lehrerausbildung gewonnen wurden.

## 1. Algorithmierung/Programmierung – Wichtig für die Allgemeinbildung!

Die Algorithmierung/Programmierung ist eines der zentralen Wirkprinzipien der Informatik. Auch im Alltag nimmt die Programmierung ständig weiter zu, so sollen z.B. Videorecorder, Anrufbeantworter, Handy, aber auch Waschmaschine oder Elektroherd programmiert werden. Bei Waschmaschine oder Herd geschieht das meist mit einfachen Schaltereinstellungen – bei Handys kann schon etwas mehr Aufwand erforderlich sein. Wichtig ist dabei stets eine Strukturierung von Abläufen.

Schon in der Schule ist deshalb eine Arbeitsmethodik zum Problemlösen zu vermitteln, die sich in die folgenden vier Schritte fassen läßt: Problemanalyse, Modellbildung, Algorithmierung/Implementierung und Modellkritik.

In den 70iger und 80iger Jahren hat der sogenannte „algorithmenorientierte Ansatz“ für den Informatikunterricht teilweise zu reinen Programmierkursen zum Lösen mathematischer Aufgabenstellungen in einer zur Zeit aktuellen Programmiersprache (z.B. Basic, Pascal) geführt. Viele Lehrer haben sich die notwendigen Kenntnisse der jeweils eingesetzten Sprache mit viel Mühe und Engagement in ihrer Freizeit und in mehr oder weniger kurzfristigen Fortbildungslehrgängen erarbeitet, ohne dabei auf Kenntnisse aus einem grundständigen Informatiklehrerstudium zurückgreifen zu können. Deshalb ist es durchaus verständlich, wenn sich Lehrer durch die immer schneller entwickelnden aktuellen Programmierumgebungen für professionelle Software für Industrie und Forschung überfordert fühlen. Dabei hat die Schule ja keinesfalls die Aufgabe, Schüler zu professionellen Programmieren auszubilden. Vielmehr geht es in der Schule um die Vermittlung von Grundkenntnissen

- zu Wirkprinzipien moderner Informatiksysteme (z.B. Algorithmierung/ Programmierung)
- zur Lösung von Problemen mit Hilfe von Informatiksystemen
- zum Umgang mit modernen Informations- und Kommunikationstechnologien

In seinem Artikeln „‘Roboter programmieren – ein Kinderspiel‘, Bewegt sich auch etwas in der Allgemeinbildung?“ schreibt J. Nievergelt dazu:

*„...Wenn Endbenutzer nicht mehr programmieren, folgt daraus, daß nur noch Computer Profis Programmierkenntnisse lernen sollen? Wir argumentieren, daß programmieren als Komponente der Allgemeinbildung zu unterrichten sei. Der abstrakte Begriff, zeitliche Abläufe streng zu spezifizieren, gehört im Zeitalter der Informationstechnik zum allgemeinen gedanklichen Rüstzeug. .... Programmieren als gedankliches Gut, frei vom Zwang unmittelbaren Nutzens, lernt man am besten in einer Spielumgebung, welche wichtige Begriffe in möglichst einfacher Gestalt einführt.“* [7]

Diese Idee, spezielle Umgebungen für das Lehren von Grundbegriffen der Algorithmierung und Programmierung zu schaffen, ist nicht neu. Verwiesen sei auf den bekannten Roboter Niki, auf Karel the Robot [9], [10] oder auf Kara, den programmierbaren Marienkäfer [11]. Aus dem gleichen Ansatz heraus wurde auch das Java-Hamster-Modell zur „spielerischen" Einführung in die objektorientierte Programmierung mit Java entwickelt [1]. In dem seit 10 Jahren an der TU Dresden durchgeführten Erweiterungsstudium für das Lehramt Informatik an Mittelschulen (Realschulen), Lehramt an Gymnasien und für das Lehramt an berufsbildenden Schulen spielten diese Modelle bisher nur eine geringe Rolle. Sie waren insbesondere in den Lehrveranstaltungen zur Didaktik der Informatik Gegenstand von Studentenvorträgen und Seminardiskussionen.

In den sich über alle drei Semester des Grundstudiums erstreckenden Vorlesungen, Übungen und Seminaren zur Algorithmierung/Programmierung werden die wichtigen Begriffe und Konzepte anhand der Sprachen Logo, PC Scheme und Pascal vermittelt. Dabei wird im 1. Semester stets mit einer Einführung in die funktionale Programmierung (mittels Logo, PC Scheme) begonnen. Anschließend kam im vergangenen Wintersemester erstmals Delphi 4.0 zum Einsatz. Beweggründe für diesen Aufbau der Lehrveranstaltungen zur Algorithmierung/ Programmierung werden in den nächsten beiden Abschnitten dargelegt. Im letzten Abschnitt werden dann einige Erfahrungen aus der bisherigen Ausbildung zusammengefaßt und ein Ausblick auf die weitere Entwicklung gegeben.

## 2. Klassische Programmierung im Kleinen versus Objektorientierung und Programmierung im Großen

Die Grundprinzipen der Algorithmierung/Programmierung und der verschiedenen Programmierstile wie imperative, funktionale oder logische Programmierung lassen sich relativ gut bereits anhand kleinerer Problemstellungen erfassen und umsetzen. Dabei stellen Funktionen eine besonders einfache Umsetzung der Begriffe: **E**ingabe (über Parameter), **V**erarbeitung (Berechnungsvorschrift) und **A**usgabe (Rückgabe des Funktionswertes) dar. In der ersten Vorlesung zur Algorithmierung/Programmierung werden in den Lehramtsstudiengängen an der Fakultät Informatik deshalb folgende Schwerpunkte behandelt:

- der „naive" Algorithmusbegriff und Möglichkeiten der Formulierung von Algorithmen
- Überblick über verschiedene Programmierparadigmen

Der Begriff „Algorithmus" wird in den Vorlesungen zur Theoretischen Informatik mittels der TURING-Maschine präzise definiert. Vorerst genügt ein sogenannter „naiver" Algorithmusbegriff, der folgendermaßen beschreiben wird:

*„Unter einem Algorithmus versteht man eine Verarbeitungsvorschrift, die so präzise formuliert ist, dass sie von einem mechanisch oder elektronisch arbeitenden Gerät ausgeführt werden kann. Aus der Präzision der sprachlichen Darstellung des Algorithmus muss die Abfolge der einzelnen Verarbeitungsschritte eindeutig hervorgehen. Hierbei sind Wahlmöglichkeiten zugelassen. Nur muss dann genau festliegen, wie die Auswahl einer Möglichkeit erfolgen soll."* [2]

Dieser Beschreibung des Algorithmusbegriffs liegt eine befehlsorientierte Denkweise, die in der Struktur und Arbeitsweise eines von-NEUMANN-Rechners begründet ist, zugrunde. Dazu folgendes Beispiel:

**Problem:** Es ist die kleinste Zahl von *n* natürlichen Zahlen (eines Feldes natürlicher Zahlen) zu ermitteln.

**Algorithmus:**
- Eingabe der *n* natürlichen Zahlen als Feld
- Setze: kleinstes_Element = erstes Element des Feldes
- für das 2-te bis *n*-te Element wiederhole:
  Ist das *i*-te Element ($2 \leq i \leq n$) kleiner als das kleinste_Element?
  Falls ja, dann setze: kleinstes_Element = *i*-tes Element
  Falls nein, dann tue nichts
- Gib das kleinste_Element aus

Schon diese Formulierung des Algorithmus mit Worten der Umgangssprache stellt eine Reihe von Anforderungen an das Abstraktionsvermögen eines „Programmieranfängers“, insbesondere, wenn er wenig Freude an abstraktem mathematischen Denken hat. Er muss ja ganz exakt angeben, *was das Computersystem in welcher Reihenfolge zu tun hat*.

So kann es unter Umständen leichter sein, das zu lösende Problem mittels funktionaler Zusammenhänge zu *beschreiben*. Dazu wird festgehalten:

Ein Algorithmus wird durch eine Funktion (Abbildung)

**f: E ® A**

von der Menge der zulässigen Eingangsdaten in die Menge der Ausgangsdaten (EVA-Prinzip: **E**ingabe - **V**erarbeitung - **A**usgabe) beschrieben. Dies ist in der folgenden Skizze dargestellt:

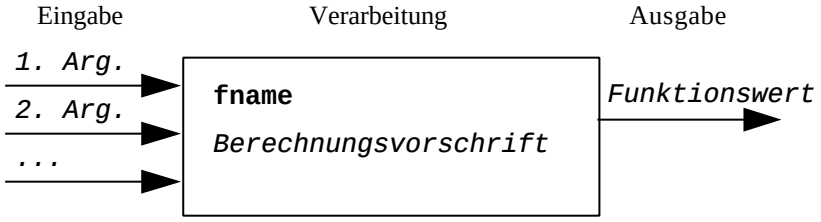



*Abbildung1: Funktion*

Betrachtet man nun obige Aufgabe unter diesem Gesichtpunkt, kann man formulieren:

**Problem:** Es ist die kleinste Zahl einer Liste von natürlichen Zahlen zu ermitteln.

**Algorithmus:** Die Funktion, die den Lösungsalgorithmus beschreibt, hat genau ein Argument (eine Liste von natürlichen Zahlen) und sie gibt die kleinste Zahl als Funktionswert zurück. Dieser Funktionswert wird folgendermaßen „berechnet“: Wenn die Liste nur ein Element hat, so ist dieses Element als Funktionswert zurückzugeben. Andernfalls ist zu prüfen, ob das 1. Element größer als das 2.Element der Liste ist. Falls ja, ist die Funktion wieder anzuwenden auf die Liste, aus der das 1. Element entfernt wurde. Falls nein, ist die Funktion wieder anzuwenden auf die Liste, die durch Entfernen des 2. Elements aus der Ausgangsliste entsteht: Die interne Abarbeitung dieser Funktion sei kurz für ein spezielles Beispiel skizziert:

```
minimum [6 2 4]
↘  minimum [2 4]
     ↘  minimum [ 2]   → 2
```

Zugegebenermaßen erscheint die verbale Formulierung auch in diesem Fall relativ kompliziert. Sie läßt sich jedoch recht einfach in eine entsprechende nutzerdefinierte Logo-Funktion umsetzen:

```
to minimum :li
 ifelse emptyp bf :li [op first :li]~
  [ifelse (first :li) > (zweites :li) [op minimum bf :li]~
                                      [op minimum ohne_zweites :li]]
end
```

Natürlich müssten noch die nutzerdefinierten Funktionen `zweites :li` und `ohne_zweites :li` geschrieben werden. Dies ist aber kein Problem. Man kann auf diese zusätzlichen Funktionen verzichten, dann wird der Quelltext etwas schlechter lesbar:

```
to mini :li
 ifelse emptyp bf :li [op first :li]~
  [ifelse (first :li) > (first bf :li) [op mini bf :li]~
                                       [op mini fput first :li bf bf :li]]
end
```

Anhand dieses Beispiels sollte der Unterschied zwischen dem imperativen Paradigma und dem funktionalen Paradigma verdeutlicht werden.

Die nachfolgende Tabelle gibt einen Überblick über die verschiedenen Programmiersprachen und die zugrundeliegenden Programmierparadigmen (vgl. [12] bzw. S. 201 in [7]).

| | **Programmiersprachen** | | |
|---|---|---|---|
| | **prozedurale Sprachen** | **deklarative Sprachen** | |
| | **imperative Sprachen** | **funktionale Sprachen** | **prädikative Sprachen** |
| **Beispiele** | ALGOL, FORTRAN, BASIC, PASCAL, C, MODULA u.a. | LISP (SCHEME), LOGO, ML, CAML, MIRANDA, HASKELL, GOFER u.a. | PROLOG, CLP |
| **Paradigma** | **imperatives Paradigma** | **funktionales Paradigma** | **logisches Paradigma** |
| **Kalkül, math. Modell** | Algorithmen- und Automatentheorie, von Neumann-Architektur, Registermaschine | λ-Kalkül (Church) (math. Formalismus zur Beschreibung von Funktionen) | Prädikatenlogik, Horn-Klauseln |
| **Programm** | in Strukturen angeordnete Folge von Befehlen (Anweisungen) | Menge von Funktions-definitionen, die mit-einander verkettet sind | Sammlung von Fakten und logischen Schlußregeln |
| **zentraler Begriff** | **Befehl** (**Anweisung**) | **Funktion** (**Ausdruck**) | **Faktum** |
| **Variablenkonzept** | Variable als Name-Wert-Paar | Variable ist Bezeichner, der an ein konkretes Objekt gebunden wird | Variablen sind im prädikativen Sinne Unbestimmte |

Allen diese „klassischen“ Programmierstilen liegt letzten Endes eine sequentielle Arbeitsweise zugrunde (Fließbandarbeit).

Die **objektorientierte Programmierung** wurde bewusst nicht in dieses Schema eingeordnet und zwar aus folgendem Grund: Die Objektorientierung beruht auf einer ganz neuen Sichtweise, die bereits in der Modellierungsphase eine Rolle spielt. Nicht umsonst spricht man von objektorientierter Analyse (OOA), objektorientiertem Entwurf (OOE) und Design (OOD) und schließlich von objektorientierter Implementierung und Programmierung (OOP) [7]. Dabei arbeitet man mit Objekten, die aus Objektklassen abgeleitet werden und über deren Eigenschaften und Methoden verfügen. Die Objekte können Botschaften miteinander austauschen und auf diese Botschaften reagieren. Dies lässt sich durchaus mit „Teamwork“ vergleichen. Bei der Programmierung der Methoden wird jedoch auf die klassische imperative oder funktionale Programmierung zurückgegriffen. Kenntnisse über algorithmische Grundstrukturen sowie über Funktionen, Prozeduren, formale und aktuelle Parameter u.a. sind also auch für die objektorientierte Programmierung unverzichtbar. Damit man „von Anfang an“ wirklich objektorientiert modellieren und programmieren kann, muss das zu lösende

Problem im allgemeinen bereits hinreichend komplex sein. Man spricht in diesem Zusammenhang auch oft von der „Programmierung im Großen“.

Aus den Unterrichtserfahrungen in einigen sächsischen Gymnasien hat sich im Laufe der letzten Jahre folgende Erkenntnis herausgebildet: Ein erster Schritt in Richtung Objektorientierung kann darin bestehen, dass man vordefinierte Objektklassen zur Gestaltung einer nutzerfreundlichen Oberfläche für die Ein- und Ausgabe verwendet und zunächst die elementaren Grundbegriffe der imperativen oder funktionalen Programmierung vermittelt und nutzt. Dies ist sowohl mit MSW Logo, Delphi, JavaScript und anderen Sprachen möglich (vgl. [14], [15]).

## 3. Ereignissteuerung in Logo – oder: Logo – eine nach wie vor aktuelle Programmierumgebung für die Schule

Mit diesem Beitrag soll keinesfalls eine Neuauflage des uralten Sprachenstreites wieder entfacht werden. Vielmehr sei auf die Ergebnisse der Arbeitsgruppe „Programmiersprachen“ des 5. Fachdidaktischen Gesprächs verwiesen. Dort wurden einige wesentliche Kriterien für die Auswahl einer Sprache für den Informatikunterricht herausgearbeitet:

*„Primär sollen den Schülerinnen und Schülern mit Hilfe von Programmiersprachen grundlegende Begriffe und Methoden der Informatik vermittelt werden. Dazu gehört u. a.*

- *Verständnis der Funktionsprinzipien von Informatiksystemen*
- *fundamentale Strukturierungsmethoden (Daten und Algorithmen)*
- *Prozedur- und Funktionskonzepte, Parametrisierungen*
- *Modularität*
- *Denken im zeitlichen Nach- und Nebeneinander und Modellierung entsprechender Prozesse*

*Zugleich sollen die Schülerinnen und Schüler Methoden des systematischen Problemlösens erlernen, die in der Informatik und im täglichen Leben eine wichtige Rolle spielen.*
*Sekundär, aber nicht unwichtig ist die Wahl der verwendeten Sprache. Als Kriterien für die Auswahl einer Sprache werden vorgeschlagen:*
*aus informatischer Sicht*

- *ausgereifte Entwicklungsumgebung*
- *Portabilität*
- *Typsicherheit*
- *Orthogonalität der Konzepte*

*aus didaktischer Sicht*

- *Verfügbarkeit (auch von zugehöriger Literatur)*
- *Altersangemessenheit der Programmiersprache*
- *Zeitvolumen, das zur Verfügung steht*
- *Vorwissen der Schülerinnen und Schüler*

*Aus didaktischer Sicht ist der industrielle Verbreitungsgrad einer Programmiersprache allein weder ein Argument für noch gegen die Wahl der Sprache für den Einsatz in der Schule.“ ...*
*„Es wird empfohlen, dass die Schülerinnen und Schüler im Grundfach Informatik im Verlauf mehrerer Jahrgangsstufen mindestens zwei verschiedenene Programmierparadigmen kennenlernen. Für das Leistungsfach Informatik ist die Beschäftigung mit den verschiedenen Programmierparadigmen unbedingt erforderlich.“* [13]

Diese Kriterien erfüllt MSW Logo sicher nicht in allen einzelnen Punkten. Dennoch eignet sich diese Programmierumgebung sehr gut für die Umsetzung eines Gesamtkonzeptes der informatischen Bildung:

- Bereits in der Grundschule können die Schüler mit Hilfe sehr weniger Grundworte von der Kommandoebene aus vielfältige Grafiken zeichnen (Stichwort: entdeckendes Lernen). Logo wurde ja genau mit diesem Ziel, Kinder auf spielerische Weise an die Grundideen der Programmierung heranzuführen, entwickelt.
- In der Sekundarstufe I kann man auch mit Logo durchaus algorithmische Grundstrukturen wie Folge, Schleife und Verzweigung behandeln, wenn man sich auf entsprechende Prozeduren zum Zeichnen von Grafiken beschränkt. Beispiele für diese Herangehensweise findet man im Profilband des Lehrbuchs „Informatik und Alltag“ [5], welches speziell für den Informatikunterricht an sächsichen Mittelschulen geschrieben wurde.
- Ebenso könnte man in der Sekundarstufe I bereits die elementaren Grundbegriffe der funktionalen Programmierung wie Abstraktion, Applikation und Komposition einführen und einfache Probleme der Listenverarbeitung (s. obiges Beispiel: Minimum einer Liste von Zahlen) bearbeiten. Dabei soll unter obigen Begriffen folgendes verstanden werden:
  **Abstraktion:** Man kann
  - eine Berechnungsvorschrift an einen Funktionsnamen
  - eine Verarbeitungsvorschrift an einen Prozedurnamen
  - den Wert eines Ausdruckes an einen Namen
  - mehrere Eigenschaften (bestehend aus Namen und Wert) an einen Namen (einer Eigenschaftsliste)

  binden.
  *Beispiel:*
  ```
  to mittelwert :a :b
   output (:a + :b)/2
  end
  ```
  **Applikation:** Die bei einer Funktions- oder Prozedurdefinition angegebenen Argumente sind formale Parameter, die beim Aufruf der Funktion oder Prozedur durch aktuelle Parameter (auswertbare Ausdrücke) ersetzt werden müssen. Die Funktion oder Prozedur wird auf diese aktuellen Parameter angewandt:
  *Beispiel:* `show mittelwert 3 4`
  **Komposition:** Funktionen können miteinander verkettet werden, d.h. Funktionswerte können selbst Argumente anderer Funktionen sein:
  *Beispiel:* `show mittelwert sqrt 9 sqrt 16`
- In der Sekundarstufe II schließlich kann man anspruchsvollere Themen behandeln wie
  - Definition und Durchsuchen von Bäumen (Präorder, Inorder und Postorder-Durchlauf für binäre Bäume, Tiefen- und Breitensuche)
  - Erstellung kleiner Datenbanken mittels Eigenschaftslisten und Definition einfacher Abfragen
  - Zeichnen fraktaler Grafiken.

Weitere wesentliche Vorteile von MSW Logo sind:

- Es ist, wie die meisten funktionalen Sprachen, ein Interpretersystem. Man kann auf der Kommandoebene hervorragend experimentieren, bevor man im Editor nutzerdefinierte Prozeduren zum Zeichnen von komplexen Grafiken schreibt. Dabei kommt man für den Anfangsunterricht mit drei Grundprozeduren aus: `forward <.>`, `right <.>`, `clearscreen`.
- Mit den arithmetischen Grundfunktionen und weiteren mathematischen Standardfunktion kann man schnell einfache Berechnungen von der Kommandoebene aus aufrufen.
- Für einfache Probleme der Listenverarbeitung kommt man ebenfalls mit wenigen Grundprozeduren aus: `emptyp <.>`, `first <.>`, `butfirst <.>`, `fput <.> <.>`.

- Für bedingte Ausdrücke und zur Programmierung von Alternativen steht zur Verfügung: `ifelse <bedingung> [...] [...].`
- Die Rekursion als wichtige Strategie zum Lösen von Problemen ergibt sich fast „von selbst“ (s. obiges Beispiel: Minimum einer Liste von Zahlen). Dabei stellt MSW Logo mit dem TRACE-Modus ein hervorragendes Werkzeug zur Veranschaulichung der endständigen Rekursion (Iteration) und der echten Rekursion zur Verfügung.
- MSW Logo wird in den USA nach wie vor für ältere und neuere PC (16-Bit-Rechner, 32-Bit-Rechner) gepflegt.
- MSW Logo integriert auch moderne Programmierungstechniken: Erzeugen animierter Gif-Grafiken, Einbindung von Sound, Netzprogrammierung u.a.

Außerdem stellt MSW Logo eine Reihe von Grundprozeduren und Grundfunktionen zur Erzegung von Instanzen (Objekten) vordefinierter Objektklassen wie Fenster, Labels, Buttons usw. zur Verfügung:

Zum Erzeugen eines neuen Fensters:
**`windowcreate`** `<eltern> <name> <beschriftung> <x-pos> <y-pos> <breite> <hoehe> <ereignis>`
Zum Löschen eines Fensters:
**`windowdelete`** `<name>`
Zum Erzeugen eines Labels:
**`staticcreate`** `<eltern> <name> <beschriftung> <x-pos> <y-pos> <breite> <hoehe>`
Zum Erzeugen eines Ein- und Ausgabefensters
**`comboboxcreate`** `<eltern> <name> <x-pos> <y-pos> <breite> <hoehe>`
Zum Lesen aus einem Ein- und Ausgabefenster
**`comboboxgettex`**`t <name>`
(Die Grundfunktion `comboboxgettext <..>` gibr den Inhalt der Box als Liste zurück.)
Zur Ausgabe in ein Ein- und Ausgabefenster
**`comboboxsettext`** `<name> <inhalt>`
Zum Erzeugen eines Buttons:
**`buttoncreate`** `<eltern> <name> <beschriftung> <x-pos> <y-pos> <breite> <hoehe> <ereignis>`

Leider ist zwar die pixelgenaue Angabe der linken oberen Ecke und der Höhe und Breite jedes Objekts für die Plazierung im Fenster anfangs etwas mühsam. Dennoch ist dies insgesamt eine wesentliche Erleichterung für die Ein- und Ausgabe, da man beim Aufruf jeder Logo-Funktion oder Logo-Prozedur die Anzahl und Reihenfolge der Argumente beachten muss.

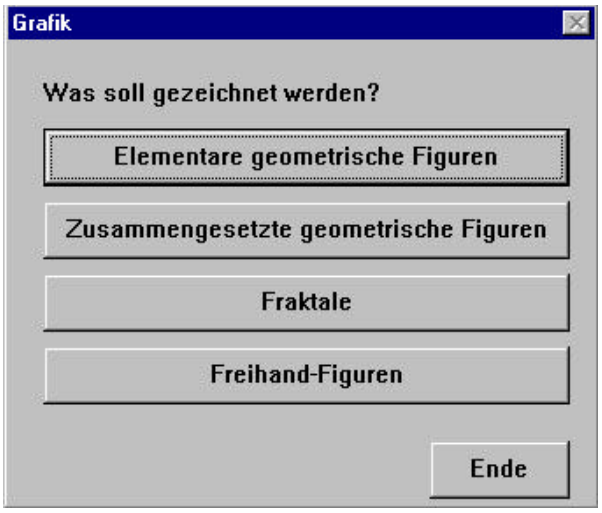
## 4. Erfahrungen aus der Ausbildung von Lehrern an der TU Dresden

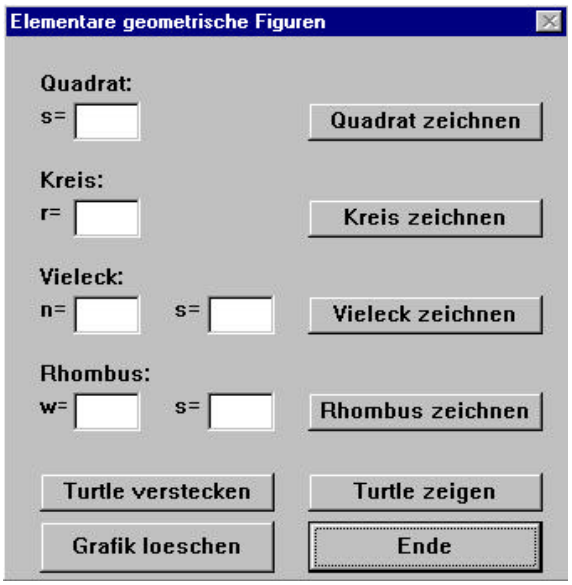
Seit 1991 nahmen jährlich ca. 50 bereits im Beruf stehende Lehrer unterschiedlicher Fächer ein berufsbegleitendes Studium für Informatiklehrer auf. Für dieses Studium stehen laut einer entsprechenden Verordnung des Sächsischen Staatsministeriums für Kultus insgesamt 40 Semesterwochenstunden für das Lehramt an Mittelschulen und 60 Semesterwochenstunden für das Lehramt an Gymnasien und für das Lehramt an berufsbildenden Schulen zur Verfügung. Dabei absolvieren alle Studenten ein gemeinsames Grundstudium im Umfang von 30 Semesterwochenstunden. Ein Drittel dieses Zeitvolumens steht für das Themengebiet „Algorithmierung/Programmierung“ zur Verfügung. Wie bereits erwähnt, wird zunächst mit einer Einführung in die funktionale Programmierung anhand der Sprachen Logo und PC Scheme begonnen. Dieser Vorlesungsteil ist folgendermaßen gegliedert:

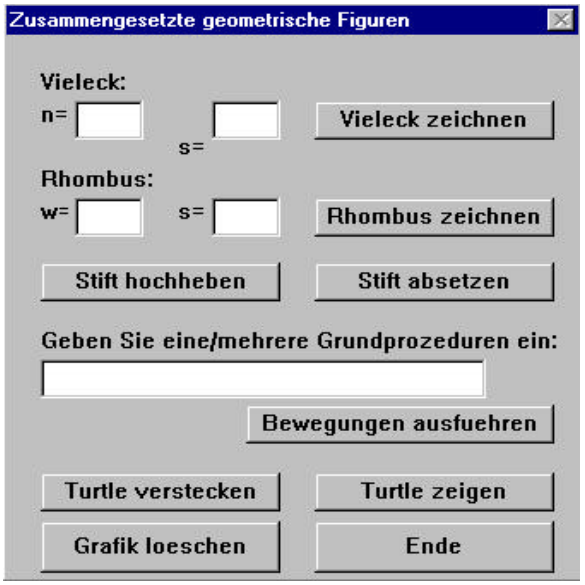
1. Einführung in die Algorithmierung/Programmierung (Der „naive“ Algorithmusbegriff, Programmierparadigmen im Überblick)
2. Einführung in die funktionale Programmierung mit Logo und PC Scheme (Grundbegriffe: Abstraktion, Applikation, Komposition; Atome und Listen; Grundfunktionen, -prozeduren und nutzerdefinierte Funktionen, -prozeduren; Prädikate und bedingte Ausdrücke)
3. Rekursion als Strategie zum Problemlösen (Iteration – Rekursion; Lineare Rekursion; Baumrekursion)
4. Rekursive Datenstrukturen: Listen und Bäume (Grundfunktionen zur Arbeit mit Listen; Lineare Listen; Bäume)
5. Eigenschaftslisten (Grundfunktionen; Anwendungsbeispiele)
6. Globale Bindungen und das Referenzprinzip (Grundfunktionen; Anwendungsbeispiele)
7. Gestaltung nutzerfreundlicher Oberflächen mittels vordefinierter Objektklassen
8. Grafik in Logo und PC Scheme

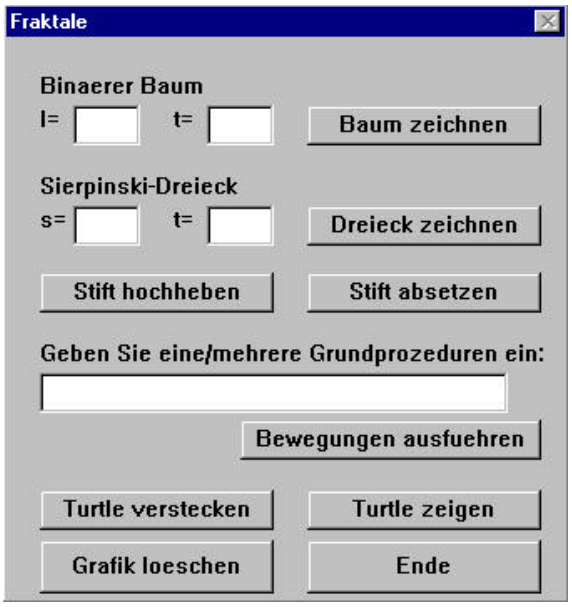
Die in der Vorlesung behandelten Beispiele werden den Studenten als Quelltext in PC Scheme und in Logo zur Verfügung gestellt. In den praktischen Übungen bevorzugen die Lehrer meist das Logo-System. Jedoch wurde die Grafik ganz bewusst an das Ende der Vorlesungsreihe gesetzt. Dieser Einstieg mit der funktionalen Programmierung hat u.a. auch den Vorteil, dass in aller Regel auch denjenigen Studenten, die bereits einige eigene Programmiererfahrungen besitzen, eine neue Sichtweise vermittelt wird. Außerdem steht MSW Logo frei zur Verfügung und kann auch von Anfängern ohne Kosten und ohne Schwierigkeiten auf dem eigenen PC installiert werden.

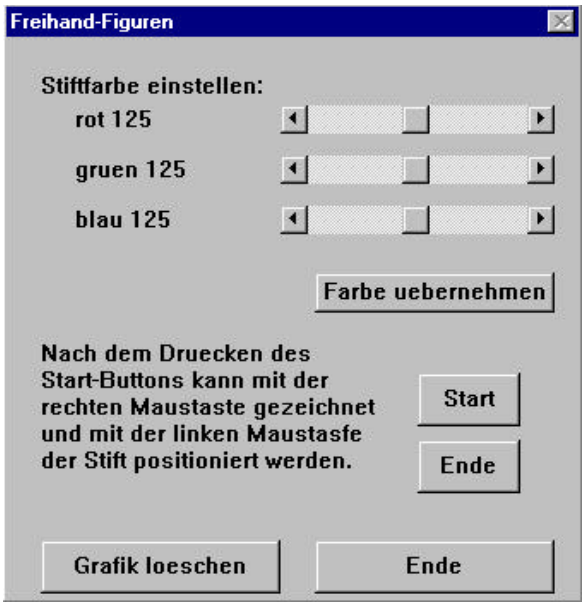
Anschließend wurde bisher mit der imperativen Programmierung in PASCAL fortgesetzt. Im vergangenen Semester wurde erstmalig auch die ereignisgesteuerte Programmierung in Logo und die Arbeit mit vordefinierten Objektklassen zur Erstellung einer nutzerfreundlichen Oberfläche behandelt. Mit viel Freude und Engagement lösten die Studenten nachfolgende Praktikumsaufgabe:
*„Nach Aufruf der Prozedur „Start“ soll das Commander-Fenster als Icon abgelegt und das unten abgebildete Startfenster erzeugt werden: Das Anklicken der Buttons ruft jeweils ein weiteres der abgebildeten Fenster auf. Auch die Buttons dieser Fenster sind durch entsprechende Prozeduren und Funktionen zu hinterlegen.*






Diese Einführung in die Arbeit mit vordefinierten Objektklassen sollte den Studenten den Umstieg auf Delphi erleichtern. Trotzdem hatten insbesondere diejenigen Lehrer, die vor dem Studium nur wenig Erfahrungen in der Computernutzung gesammelt haben, mit der komplexen Oberfläche von Delphi größere Probleme. Diese Probleme behinderten die zügige Behandlung der Grundbegriffe der imperativen Programmierung. Gleiche Erfahrungen berichtete während der schulpraktischen Übungen übrigens auch ein Gymnasiallehrer. Dieser hatte in zwei Klassen etwa gleicher Leistungsstärke einen Einführungskurs in die imperative Programmierung gehalten, in der einen Klasse mit Turbo Pascal und in der anderen Klasse unter Delphi. Aufgrund der Komplexität von Delphi kam er mit der Behandlung der algorithmischen Grundstrukturen in Pascal wesentlich schneller voran. Außerdem verleiten die vielen Gestaltungsmöglichkeiten in Delphi zu sehr zum Experimentieren an der Oberfläche für die Ein- und Ausgabe. Diese Erfahrungen sprechen somit durchaus für eine spezielle Programmierumgebung für das Lehren von Grundbegriffen und Konzepten der Algorithmierung/Programmierung wie die bereits im 1. Abschnitt erwähnten Systeme Karel,

Kara oder das Java-Hamster-Modell. Ein besonderer Vorzug des Java-Hamster-Modells ist dabei, dass sich dieses Modell an den Sprachkonzepten von Java orientiert.

Im Hauptstudium der Mittelschullehrer bleibt leider nur noch wenig Zeit für die Algorithmierung/Programmierung. Im Sommersemester 2000 wurde erstmalig im Rahmen einer Vertiefungsrichtung ein Kurs zur Programmierung für das Web (Sprache: JavaScript) angeboten, der sehr großen Zuspruch fand. Da dieser Jahrgang allerdings im bisherigen Studium noch nichts von ereignisgesteuerter Programmierung oder gar objektorientierter Programmierung gehört hatte, musste entsprechend viel Zeit in die Vermittlung der Grundbegriffe investiert werden. Die im August dieses Jahres vom Senat der TU Dresden bestätigte Fachstudienordnung für den grundständigen Studiengang Lehramt an Mittelschulen, Fach Informatik, sieht insgesamt 57 Semesterwochenstunden vor. Dabei bleibt zwar der Zeitfonds für die Algorithmierung/Programmierung im Grundstudium unverändert, aber im Hauptstudium stehen für die Vertiefungsrichtungen statt bisher 4 dann 8 Semesterwochenstunden zur Verfügung.

Im Hauptstudium der Gymnasial- und Berufsschullehrer hingegen wird seit Anfang an sowohl logische Programmierung mit Prolog als auch objektorientierte Programmierung (bisher mit Borland Pascal, ab Sommersemester 2000 mit Objekt Pascal unter Delphi) als Pflichtlehrveranstaltung angeboten. Außerdem muss jeder Student im 5. und 6. Semester eine der Vertiefungsrichtungen (im Umfang von insgesamt 8 Semesterwochenstunden) belegen:

- Datenbanken – Rechnernetze
- Algorithmierung/Programmierung – Programmierparadigmen
- Modellbildung und Simulation
- Kryptologie

Bisher wurden in der Vertiefungsrichtung zur Programmierung u.a. Lehrveranstaltungen zur funktionalen Programmierung mit PC Scheme, zu Algorithmen der Künstlichen Intelligenz (mit Prolog) und auch zu Numerischen Algorithmen angeboten. In der letztgenannten Lehrveranstaltung standen die Algorithmen zur Lösung numerischer Probleme im Vordergrund und die Umsetzung wurde parallel mit ganz verschiedenen Werkzeugen wie Pascal, Logo, PC Scheme aber auch Excel, Mathematica und Maple erprobt. Zum Einsatz von Java in der Lehramtsausbildung liegen bisher noch keine Erfahrungen vor. Ein entsprechender Kurs für Magisterstudenten offenbarte einige Probleme, die in der raschen Entwicklung von Java selbst begründet sind. Aus diesem Grund wird im kommenden Semester auch im Magisterstudiengang mit JavaScript statt Java gearbeitet werden. Die im August dieses Jahres bestätigte Fachstudienordnung für den grundständigen Studiengang Lehramt an Gymnasien und Lehramt an berufsbildenden Schulen sieht insgesamt statt bisher 60 nun 71 Semesterwochenstunden für die Ausbildung vor, so kann auch in diesem Studiengang mehr Zeit für das Themengebiet Algorithmierung/Programmierung bzw. Softwaretechnik bereitgestellt werden In diesem Zusammenhang ist auch eine engere Kopplung mit Lehrveranstaltungen aus dem Diplomstudiengang Informatik vorgesehen.

**Literatur:**

[1] Boles, D., Programmieren spielend gelernt mit dem Java-Hamster-Modell, B.G. Teubner Verlag, Stuttgart, Leipzig, 1999, s. auch: http://www-is.informatik.uni-oldenburg.de/~dibo/hamster/simulator.html, ISBN 3-519-02297-4

[2] Claus, V., Schwill, A., Duden Informatik, Dudenverlag Mannheim, Leipzig, Wien, Zürich, 1993, ISBN 3-411-05232-5

[3] Ettinger, W., Findeis, W., Freiberger, U., Großmann, R., Künzel, E., Worg, R., Handreichung für den Informatikunterricht am Gymnasium, Teil 1: Ziele, konzepte, Vorschläge für den Unterricht, Staatsinstitut für Schulpädagogik und Bildungsforschung München, März 1997

[4] Friedrich, S. (Hrg)., Informatik und Alltag, Basisband, Dümmlerbuch 4987, Ferd. Dümmlers Verlag 1998, ISBN 3-427-49873-5

[5] Friedrich, S. (Hrg)., Informatik und Alltag, Profilband, Dümmlerbuch 4991, Ferd. Dümmlers Verlag, 1997, ISBN 3-427-49911-1

[6] Friedrich, S. (Hrg)., Handreichung für Lehrerinnen und Lehrer zu Informatik und Alltag, Profilband, Dümmlerbuch 4992, Ferd. Dümmlers Verlag, 1997, ISBN 3-427-49921-9

[7] Horn, Ch., Kerner, I.O., Lehr- und Übungsbuch Informatik, Band 3: Praktische Informatik, Fachbuchverlag Leipzig im Carl Hanser Verlag München, 1997, ISBN 3-446-18699-9

[8] Nievergelt, J., „Roboter programmieren“ – ein Kinderspiel, Bewegt sich auch etwas in der Allgemeinbildung? in: Informatik-Spektrum 22 (1999) 5, S. 364-375

[9] Pattis, R.E., Karel the Robot – A Gentle Introduction to the Art of Programming, Wiley, New York 1981,

[10] Patties, R., Robert, J., Stehlik, M, Karel the Robot: A Gentle Intorduction to the Art of Programming, Wiley, New York 1995, s. auch: http://www.mtsu.edu/~untch/karel/index.html

[11] Reichert, R., Ein spielerischer Einstieg ins Programmieren – Kara, der programmierbare Marienkäfer, Diplomarbeit ETH Zürich, 1999, s. auch: http://www.educeth.ch/informatik/karatojava/kara/

[12] Schwill, A., Programmierstile, in: LOG IN 13 (1993), Heft 4, S. 10 - 19

[13] Die Ergebnisse der Diskussion in der Arbeitsgruppe „Programmiersprachen“ findet man auf der Webseite: http://koenigstein.inf.tu-dresden.de/99/sprachen/index.html

[14] Kurse zur Einführung in die imperative Programmierung (mit Delphi) in der Schule findet man unter: http://www.plauener.de/~lessing/delphi/, http://home.germany.net/101-285181/delphi.htmund http://rcswww.urz.tu-dresden.de/~tj398838/delphi.html

[15] Folien zum Vortrag „Ereignisgesteuerte Programmierung mit MSW LOGO“ findet man unter: http://hyperwave.inf.tu-dresden.de/absolvententreffen/06/index.htm

**Anschrift:**
Dr. Bettina Timmermann
TU Dresden, Fakultät Informatik
Institut für Software- und Multimediatechnik
Arbeitsgruppe Didaktik der Informatik/Lehrerbildung - L
01062 Dresden
e-mail: bettina.timmermann@inf.tu-dresden.de