Intuitive Modelle in der Informatik

Michael Weigend

Holzkamp-Gesamtschule
Willy-Brandt-Str. 2
D-58452 Witten
michael.weigend@fernuni-hagen.de

Abstract: Bei der Analyse und Kreation von Programmtexten verwenden wir Intuitionen, die nicht durch die Programmiersprache determiniert sind, sondern auf Alltagserfahrungen gründen. In diesem Beitrag werden intuitive Modelle zur Benennung von Objekten diskutiert und mit der "Python Visual Sandbox" ein methodischer Ansatz zu ihrer Erforschung vorgestellt.

1 Was sind intuitive Modelle?

Wir verwenden intuitive Modelle der Informatik, wenn wir versuchen, einen Programmtext zu verstehen oder jemandem zu erklären. Wenn wir nach der Lösung eines Problems suchen und plötzlich vor unserem geistigen Auge die Gestalt einer algorithmischen Idee auftaucht („So müsste es eigentlich klappen!“), so ist das auch ein intuitives Modell. Typische Beispiele für intuitive Modelle aus der Informatik sind folgende Gedanken:

- Objekte kann man benennen und über ihren Namen ansprechen.
- Eine Liste ist aufsteigend sortiert, wenn nirgendwo der linke Nachbar größer ist.

Nach Efraim Fischbein [Fi87] ist eine Intuition eine gedankliche Vorstellung, die insbesondere folgende Merkmale aufweist: Sie ist selbstevident, das heißt, sie ist unmittelbar einleuchtend und bedarf keiner weiteren Erklärung. Eine Intuition hat Gestalt-Charakter. Sie erscheint gewiss und wird deshalb als Grundlage für Entscheidungen und Handeln verwendet. Menschen sind so sicher, dass ihre Intuitionen stimmen, dass sie bereit sind, das Risiko persönlicher Nachteile einzugehen, wenn sie einer Intuition folgen. Intuitionen sind dauerhafte Vorstellungen. Oft werden sie bereits in der frühen Kindheit gelernt und sind fester Teil der Persönlichkeit. Sie haben etwas Zwanghaftes. Nicht selten sind intuitive Modelle unbewusst und bestimmen unser Denken und Handeln ohne dass wir es merken.

Wir verwenden hier den Begriff „intuitives Modell“ an Stelle von Intuition, um den Modellcharakter deutlicher zu betonen. Es geht darum, Aspekte der Wirklichkeit gedanklich abzubilden. Andrea diSessa beschreibt besonders grundlegende intuitive Modelle, die er phänomenologische Primitive oder kurz p-prims nennt. Eine solche Intuition ist z. B. das Konzept des Widerstandes, das zur Erklärung mechanischer, elektrischer oder psychologischer Phänomene Anwendung findet. P-Prims basieren in hohem Maße auf persönlichen sensorischen Erfahrungen und sind in der Regel so fundamental, dass man sie kaum in Worte kleiden kann [Se01].

2 Intuitive Modelle sichtbar machen – die Python Visual Sandbox

Wie findet man heraus, welche intuitiven Modelle Informatik Schüler/innen verwenden? Ein Ansatz sind „Teach back"-Befragungen. Dabei werden die Teilnehmer aufgefordert, in die Lehrerrolle zu schlüpfen und bestimmte Sachverhalte mit Hilfe von selbst angefertigten Texten und Bildern zu erklären [Ve94]. Eine zweite Methode sind individuelle Interviews (z. B. [Ch96], [Ch01]), bei denen verborgene Vorstellungen durch eine geschickte Gesprächsführung ans Licht geholt werden. Drittens gibt es Untersuchungen, in denen aus beobachtetem Verhalten (Bevorzugung bestimmter Programmkonstrukte, Fehler etc.) auf die interne Verwendung bestimmter mentaler Konzepte rückgeschlossen wird. In diesem Fall ist das vermutete Konzept inhaltlich das Produkt des Untersuchenden. Es besteht dann natürlich die Möglichkeit, dass die Interpretation des Experimentators falsch ist und in Wahrheit andere interne Überlegungen zu dem beobachteten Verhalten geführt haben.

Ein anderer Weg der Erforschung intuitiver Modelle wird mit der Python Visual Sandbox (PVS) beschritten. Die PVS ist eine über das Internet erreichbare Sammlung von Spielen und Experimentierumgebungen zur Veranschaulichung und zum Ausprobieren vorgegebener oder selbst geschriebener Python-Programme. Sie enthält über hundert Animationen, kleine Filme mit Grafiken und textuellen Elementen, die unterschiedliche intuitive Modelle zur Arbeitsweise von Programmen verwenden. Wer mit der PVS spielt, setzt in verschiedenen Kontexten und auf verschiedene Weise diese Animationen (und die enthaltenen Intuitionen) mit formalem Programmtext in Beziehung.

Eine Gruppe der PVS-Applikationen heißt *Python Visual*. Hier wird ein kurzer Programmtext vorgegeben – eventuell nur eine einzige Anweisung. Der Spieler betrachtet verschiedene Modelle und beantwortet verschiedene Fragen. Beispiel: Welche Animation würden Sie verwenden, wenn Sie jemandem die Arbeitsweise des Programms erklären müssten?

Eine Weiterentwicklung ist das *Python Quiz*. Hier analysieren die Spieler einen Programmtext Zeile für Zeile und entscheiden, ob ein Modell zu der jeweiligen Anweisung passt oder nicht. Dabei haben sie die Möglichkeit, Punkte zu setzen, die sie gewinnen oder verlieren können. Auf diese Weise kann ein Beobachter abschätzen, wie sicher sich ein Spieler bei seiner Beurteilung ist.

Ein *Python Puzzle* ist eine Art Editor, bei dem man aus vorgegebenen Programmzeilen („Bausteine") durch Bewegen mit der Maus einen Programmtext zusammensetzt (Abbildung 1). Das „gepuzzelte" Programm kann jederzeit getestet werden. In einem Ausgabefenster wird das Ergebnis des Programmlaufs dargestellt. Die Aufgabe ist, eine Funktion zu definieren (z. B. Sortieren einer Liste). Erst wenn die Funktion das erwartete Ergebnis liefert, gibt es Punkte und die nächste Aufgabe erscheint. Die intuitiven Modelle werden hier in Form von "Tipps" dargeboten, die ein Spieler abrufen kann, wenn er oder sie nicht weiter kommt. Im Hindergrund sammelt die PVS verschiedene Daten der Sitzung und speichert sie zur späteren Auswertung auf dem Server. Registriert werden z. B. Bewertungen von Tipps, Reaktionszeiten und Fehler.

Abbildung 1: Screenshot aus einem Python Puzzle. Rechts gibt es einen Vorrat an "Bausteinen", die auf der Arbeitsfläche links zu einem Programmtext zusammengebaut werden können.

3 Intuitive Modelle zur Benennung von Objekten

Wir diskutieren nun einige intuitive Modelle, die sich um die Benennung von Objekten drehen. Um Missverständnisse zu vermeiden, sei darauf hingewiesen, dass wir hier den Begriff *Objekt* in zwei Bedeutungen verwenden: Erstens in einem allgemeinen Sinne als Entität in einer realen oder gedanklichen Welt und zweitens in der programmtechnischen Bedeutung (Einheit von zusammengehörigen Daten und auf diesen arbeitenden Operationen).

Namen für Objekte der Realwelt (z. B. Gegenstände, Personen, Ereignisse) werden in früher Kindheit gelernt. Sie ermöglichen die interne gedankliche Repräsentation der externen Welt – nach Piaget der Beginn der Intelligenz. Namen haben verschiedene Funktionen. Sie identifizieren ein Objekt, ermöglichen den Zugriff (Adressierungsfunktion), bringen die Funktion des Objektes innerhalb eines Kontextes zum Ausdruck und repräsentieren somit auch Beziehungen zwischen Objekten. Namen können explizit, aber auch implizit sein. So kann man eine Stecknadel auf einer Landkarte als Name für einen Ort auffassen.

3.1 Variablennamen

In Computerprogrammen werden Namen im Zusammenhang mit Variablen verwendet. Eine bekannte Intuition für Variablen ist die Vorstellung eines Behälters für Daten (Behältermodell). Der Behälter – z. B. eine Schachtel - ist mit einem Etikett versehen, das den Variablennamen trägt. Eine Zuweisung der Form

```
x = 1
```

wird so interpretiert, dass der Behälter mit Namen (Etikett) `x` mit einem neuen Inhalt,

einer Repräsentation der Zahl `1` gefüllt wird. Der vorige Inhalt wird dabei vernichtet.

Ein alternatives intuitives Modell sieht eine Zuweisung als Benennung. Der Variablenname kann als Name eines Objektes interpretiert werden (Namenmodell). Im obigen Beispiel wird der Zahl `1` Name `x` zugeordnet. Anschaulich kann man sich eine Benennung so vorstellen: Man zeichnet einen Pfeil von dem Namen zum Objekt oder etikettiert ein Objekt mit einem Zettel, das einen Namen trägt.

Was sind die Unterschiede zwischen beiden intuitiven Modellen? Ein Objekt kann erst benannt werden, wenn es existiert. Dagegen kann ein Behälter auch leer sein. Insofern unterstützen nicht-typisierende Programmiersprachen, wie z. B. Python, eher das Namensmodell. Denn erst mit einer Zuweisung (= Benennung) wird ein neuer Name eingeführt. Dagegen kann man sich eine Variablendeklaration bei Java oder Pascal so vorstellen, dass ein zunächst leerer Behälter bereitgestellt wird, der erst später mit Inhalt gefüllt wird.

Gravierende Unterschiede werden sichtbar, wenn man folgende Anweisungsfolge interpretiert.

```
x = 1
y = x
```

Im Namenmodell erhält das Objekt `1` nun einen zweiten Namen, nämlich `y`. Das entspricht vollkommen dem Alltagsgebrauch von Namen. Für ein und dasselbe Objekt der Realität werden häufig verschiedene Namen verwendet. Wendet man das Behältermodell korrekt an, so ergibt sich folgende Interpretation: Der Inhalt der Variablen (= Behälter) `x` wird kopiert und in der Variablen `y` gespeichert.

Abbildung 2: Modelle für Mehrfachnamen

Bei unveränderbaren Objekten, wie z. B. Zahlen, ist diese Vorstellung unproblematisch. Schwierig wird es, wenn es sich um änderbare Objekte handelt, wie z. B. Listen (Python). Solche Objekte können ihren internen Zustand ändern. Betrachten wir die folgende Python-Anweisungsfolge:

```
s = [1, 1, 1]
t = s
s[0] = 5
print t
```

Die Ausgabe auf dem Bildschirm lautet `[5, 1, 1]` und nicht etwa `[1, 1, 1]`. Dieses Verhalten lässt sich mit dem Namenmodell leicht erklären. Bei der Zuweisung `t=s` erhält das Listenobjekt einen zweiten Namen, nämlich `t` (Abbildung 3 links). In der Zuweisung `s[0]=5` wird das erste Element des Listenobjektes an die Zahl `5` gebunden. Das Listenobjekt hat seine Identität behalten, so dass `t` der Name des inzwischen geänderten Objektes ist.

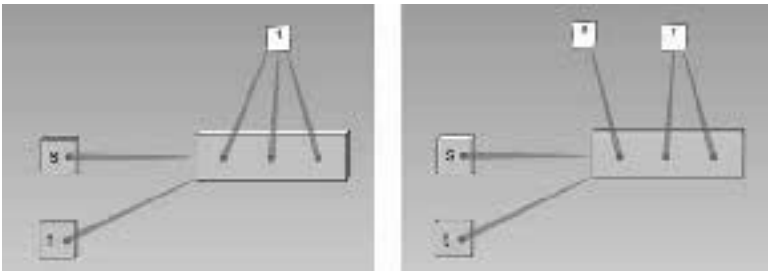

Abbildung. 3: Konsistentes Zeigermodell für die Veränderung einer Liste mit zwei Namen

Verwendet man das Behältermodell, so muss die Zuweisung `t=s` als eine Art Kopiervorgang interpretiert werden. Es entsteht ein neuer Behälter mit Etikett `t`, der den gleichen Inhalt wie s hat (Abbildung 4 links). Allerdings ist der Behälter `t` keine Kopie im üblichen Sinne. Stattdessen handelt es sich um zwei Erscheinungen des gleichen Objektes. Jede Veränderung von `s` – wie z. B. die Neubelegung der ersten Kammer mit der Zahl `5` - wird auf magische Weise ebenfalls mit `t` ausgeführt (Abbildung 4 rechts).

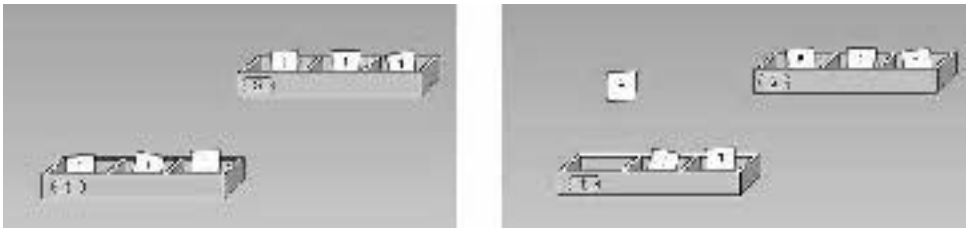

Abbildung 4: Behältermodell für die Veränderung einer Liste mit zwei Namen (Erscheinungsmodell).

3.2 Mischung von Namensmodellen – die Frage der Konsistenz

Abbildung 5 zeigt Screenshots aus verschiedenen Animationen, die die Arbeitsweise einer Iteration der folgenden Form (Python) modellieren:

```
for (n, a) in personen:
    tue etwas
```

Das erste Modell verwendet konsequent das Namensmodell, wobei Namen durch Zeiger dargestellt werden. In der zweiten Animation werden alle Variablen durch Behälter dar-

gestellt. Nach und nach werden Kopien der Daten aus dem Behälter `personen` herausgenommen und in die Behälter `n` und `a` gelegt. Die dritte Animation schließlich verwendet eine Kombination aus Zeigern und Behältern. Die Zeiger `n` und `a` „wandern“ über den Listenbehälter und zeigen nach und nach auf die Zettel in den Fächern.

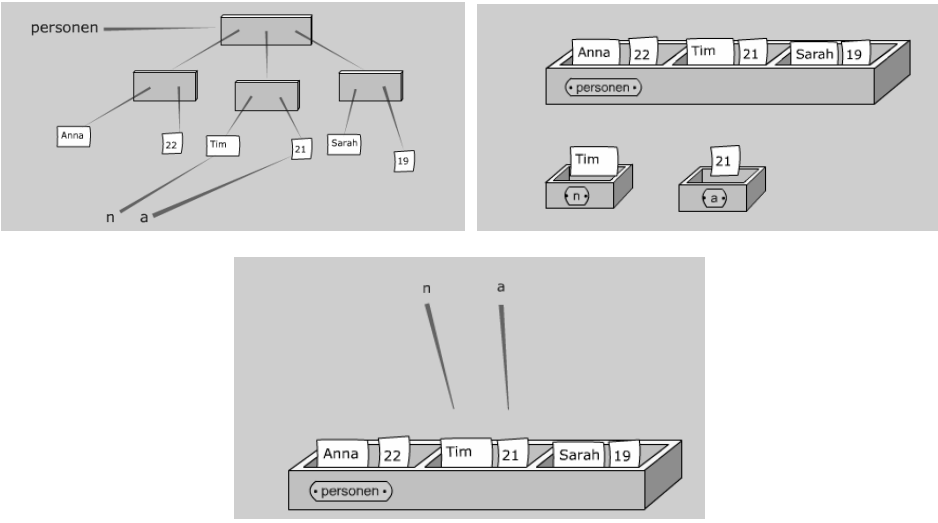

Abbildung 5: Modelle für eine Iteration über eine Liste von Paaren

Informale Befragungen von Schülern zeigen, dass trotz seiner Inkonsistenz häufig gerade das dritte Modell zur Beschreibung der Iteration bevorzugt wird. Der Grund könnte darin liegen, dass es ein häufig vorkommende Alltagssituation wiedergibt: Man durchsucht z. B. Bücher im Bücherregal, indem man nacheinander einen Blick auf jedes Buch wirft, ohne es zu bewegen. Das dritte Modell in Abbildung 5 repräsentiert also ein intuitives Modell, eine Metapher für einen vertrauten Ablauf. Das gilt für die anderen Modelle nicht. Konsistenz kann also im Widerspruch zu Intuitivität liegen.

Im Zusammenhang mit Benutzungsoberflächen hat Grudin (1989) beobachtet, dass es nicht immer auf die interne Konsistenz des konzeptuellen Modells ankommt. Analogien zur Realwelt sind häufig wichtiger für die Qualität eines User Interfaces hinsichtlich Erlernbarkeit und „intuitive“ Anwendbarkeit in neuen Situationen.

3.3 Wem gehört ein Name?

Wenn ich einen Menschen mit seinem Namen anspreche, um ihm eine Botschaft zukommen zu lassen, gehe ich davon aus, dass die Angesprochene den Namen kennt. Der Name, der im Personalausweis steht, wird als Eigentum des Benannten gesehen. Er ist Teil der persönlichen Daten und das Besitzrecht ist sogar durch Gesetze geschützt.

Andererseits existieren im Alltag auch Namen, die dem benannten Objekt unbekannt sind. Zum Beispiel erfinden Schüler hinter dem Rücken ihrer Lehrer Spitznamen für sie.

Diese Namen gehören den Individuen, die sie verwenden. Sie eignen sich zur Identifikation eines Objektes, nicht aber zur Kommunikation mit ihm.

Auch im Zusammenhang mit Namen in Computerprogrammen werden beide intuitiven Modelle verwendet. Die Botschaftsmetapher der Objektorientierten Programmierung unterstützt eher die Vorstellung, dass ein Objekt seinen Namen kennt. Denn es muss aus vielen kommunizierten Botschaften die herausfinden, die an es gerichtet sind.

In Modellen zur Veranschaulichung von Computerprogrammen können Namen durch Pfeile oder Etiketten visualisiert werden. Das Etikett klebt an einem Objekt und gehört ihm damit. Andererseits kann man nicht mehr erkennen, wer das Etikett angeklebt und so die Benennung vorgenommen hat. Beim Pfeil dagegen sind Name und Objekt räumlich getrennt. Der Pfeil beginnt bei einem Namen und dieser Name kann wiederum Attribut eines anderen Objektes sein. Hier wird also der Besitzer des Namens mit dargestellt.

3.4 Namen und Rollen

Namen haben Bedeutungen und sie werden in guten Programmen so gewählt, dass die Bedeutung erkennbar ist. In einem Funktions*aufruf* können als Argumente (Positionsargumente) andere Namen verwendet werden, als in der Liste der formalen Parameter in der Funktions*definition*.

```
def  quadrat (n):
   return n*n

seitenlaenge = input()
flaeche = quadrat (seitenlaenge)
```

Mehrfache Namen kommen auch im Alltag vor ("Mama", "Frau Müller", "Monika"). Damit wird meist zum Ausdruck gebracht, dass ein und dasselbe Objekt verschiedene Rollen spielt. Offenbar ist das Konzept des Namens eng mit dem Rollenkonzept verbunden, das heißt mit der Tatsache, dass ein Objekt in verschiedene „soziale“ Kontexte eingebunden sein kann.

Im obigen Programmbeispiel wird ein Zahlenobjekt, das zur Laufzeit über die Tastatur eingegeben wird, mit zwei unterschiedlichen Namen belegt. Im Hauptprogramm repräsentiert es die Seitenlänge eines Quadrats. Innerhalb der Definition der Funktion `quadrat()` wird das gleiche Objekt durch den Namen `n` referenziert. Hier wird das Objekt nicht mehr als Seitenlänge einer Fläche gesehen, sondern als abstraktes mathematisches Objekt. Es steht in einem anderen Kontext. Es spielt eine andere Rolle. Namensgebung ist ein Teil der Modellbildung. Betrachtet man ein Programm als System von miteinander verwobenen Miniaturmodellen, so wird über Mehrfachbenennung ein und dasselbe Objekt zum Bestandteil mehrerer Miniaturmodelle.

Jorma Sajaniemi (2002) hat ein System von „Rollen“, die Variablen in einem Programm spielen können, entwickelt und gibt für jede Rolle eine Visualisierung an. Typische

Rollen, die in Programmbeispielen von Anfänger-Lehrbüchern vorkommen, sind:

- Stepper: Eine Variable, die mit einem Anfangswert initialisiert wird und die im Laufe einer Rechnung eine bestimmte Folge von Werten durchlaufen kann. Ein Zähler, der in einer Schleife inkrementiert wird, ist ein Beispiel für einen Stepper.

- Most-wanted holder: Eine Variable, die den besten bisher gefundenen Wert enthält. Zum Beispiel braucht man bei der Suche nach dem Minimum in einer Sequenz einen most-wanted-holder für das bisher gefundene kleinste Objekt.

Oft steckt hinter einer Rolle ein intuitives Modell, das Bestandteil einer Problemlösung ist. Wenn ich einen Algorithmus für die Suche nach dem Minimum in einer unsortierten Sequenz entwickle und auf die Idee komme, einen most-wanted holder zu verwenden, habe ich bereits einen Teil der Lösung gefunden.

Auf der anderen Seite sind Rollenkonzepte für Variablen „sehr dicht am Programmtext". Sajaniemi verwendet seine Visualisierungen für die (halbautomatische) Veranschaulichung von Programmen. Das Gesamtsystem aus Rollendarstellungen ist zwar eine Verständnishilfe, aber es ist zu komplex, um intuitiv zu sein. Ein intuitives Modell, das in einer zusammenhängenden Gestalt die Idee eines Algorithmus wiedergeben soll, ist oft abstrakter und deutet Variablen nur an. Abbildung 6 zeigt Screenshots verschiedener Animationen aus der Python Visual Sandbox, die die „in place"-Sortierung einer Liste nach dem Verfahren der direkten Auswahl (straight selection) veranschaulichen.

```
s = [10, 4, 1, 3]
for i in range(len(s)):
    for j in range(i+1, len(s)):
        if s [j] < s[i]:
            s[i], s[j] = s[j], s[i]
```

Die Laufvariablen `i` und `j` sind Stepper in Sajaniemis Systematik. Sie werden im ersten Modell entsprechend seinem Vorschlag durch Papierstreifen mit Zahlen dargestellt, auf denen die aktuelle Zahl durch ein „Sichtfenster" hervorgehoben wird. In den beiden folgenden Animationen werden `i` und `j` durch Zeiger und Marken repräsentiert. Ihre Rolle ergibt sich allein aus dem Kontext der Animation. Die vierte Animation geht noch einen Schritt weiter. Hier wird die Existenz von Steppern nicht explizit erwähnt, sondern nur dadurch angedeutet, dass immer zwei Karten ein Stück aus der Kiste herausgezogen sind.

Die Kunst der Entwicklung eines Programms auf der Basis eines intuitiven Modells für den Lösungsansatz liegt in der Verfeinerung, d. h. Entfaltung der im Ausgangsmodell nur angedeuteten Facetten. Im letzten Beispiel muss man noch auf die Idee kommen, die Indexe der durch Herausziehen markierten Datenobjekte explizit zu benennen und entsprechende Variablen einzuführen.

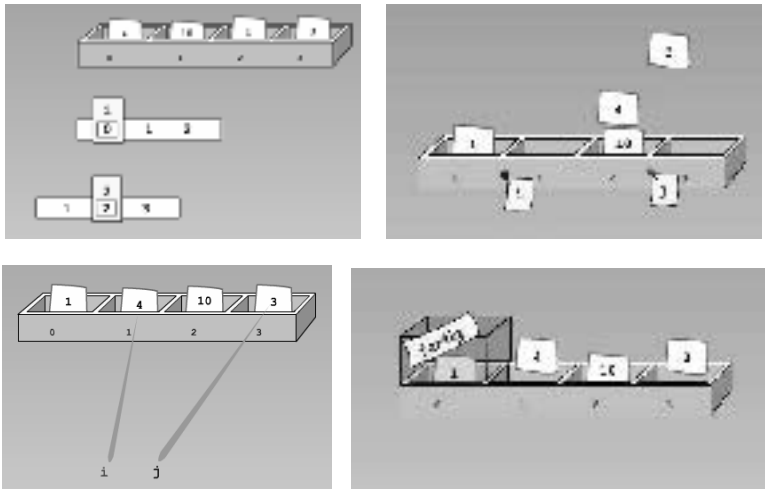

Abbildung 6: Modelle für die Sortierung einer Liste nach dem Algorithmus straight selection

4 Intuitive Modelle im Unterricht thematisieren

Intuitive Modelle können für das Verständnis der Semantik eines Programmtextes ungeeignet sein. Man bezeichnet sie dann als Fehlvorstellung (misconception). Unbewusste Fehlvorstellungen können Problemlösungen und den Erwerb neuen Wissens behindern. Deshalb wird gelegentlich gefordert, im Schulunterricht müssten Fehlvorstellungen durch bessere (wissenschaftliche) Konzepte ersetzt werden. DiSessa u. a. (Smith, diSessa und Roschelle 1993, vgl. auch diSessa 2001) kritisieren diese Auffassung. Sie argumentieren konstruktivistisch und weisen darauf hin, dass hinter Fehlvorstellungen fundamentale Konzepte stecken, die sich in vielen Lebenssituationen bewährt haben. Außerdem wird durch das Erlernen eines neuen elaborierteren Konzeptes das alte nicht vergessen. In Lernprozessen werden zuvor gelernte intuitive Modelle nicht ersetzt, sondern nur ergänzt. Das Repertoire wird größer.

Ein intuitives Modell ist niemals per se falsch. Es kann allerdings irreführend sein, wenn es in einem bestimmten Kontext falsch angewendet wird. So kann die Behälteranalogie für Variablen zu der falschen Vorstellung führen, dass bei einer Zuweisung der Form `x = y` der Inhalt des Behälters `x` in den Behälter `x` wandert und `y` anschließend leer ist. Damit ist die Behälter-Intuition aber nicht generell schlecht. In anderen Kontexten ist sie sinnvoll und leistungsfähig.

Worauf es ankommt, ist die inspirierende Kraft, aber auch die Grenzen und Fallstricke intuitiver Modelle zu kennen. Sitzungen mit der PVS, die Schülerinnen und Schüler meist zu mehreren am Rechner durchführen, sind von ausgiebigen Diskussionen der dargebotenen Visualisierungen begleitet. Derartige Übungen könnten eine sinnvolle Ergänzung eines ansonsten projektorientierten Informatikunterrichtes sein, weil sie zu einer kritischen Auseinandersetzung mit den eigenen intuitiven Modellen anregen und so zu einem vertieften Verständnis der Semantik formaler Programmtexte führen können.

Literaturverzeichnis

[Ch96] Chiu, Ming Ming: Exploring the origins, uses and interactions of students intuitions. In: Journal for Research in Mathematical Education 27 (1996), S. 478-504.

[Ch01] Chiu, Ming Ming: Using Metaphors to understand and solve arithmetic problems: Novices and experts working with negative numbers. In: Mathematical Thinking and Learning, 3.3 (2001), S. 93-124.

[Fi87] Fischbein, Efraim: Intuition in Mathematics and Science, Dordrecht Boston Lancaster Tokyo 1987.

[Gr89] Grudin, Jonathan: The Case Against User Interface Consistency. In: Communications of the ACM. Oktober 1989 Vol. 32. S. 1164-1173.

[Sa02] Sajaniemi, Jorma: Visualizing Roles of Variables to Novice Programmers. In: Kuljis, J.; Baldwin, L.; Scoble, R. (Hrsg.): Proc PPIG 14, Brunel University 2002.

[Se01] diSessa, Andrea: Changing Minds. Cambridge, London (MIT Press) 2001.

[SSR93] Smith, John P.; diSessa, Andrea A.; Roschelle, Jeremy: Misconceptions Reconceived: A Constructivist Analysis of Knowledge in Transition. In: Journal of the Learning Sciences, Vol. 3 (1993/94) Nr. 2.

[Ve94] van der Veer, Gerrit C.: Mental Models of Computer Systems: Visual Languages in the Mind. In: Tauber, M. J.; Mahling, D.E.; Arefi, F. (Hrsg.): Cognitive Aspects of Visual Languages and Visual Interfaces. Amsterdam London New York Tokyo (North-Holland) 1994, S. 3 ff.

[We04] Weigend, Michael: Objektorientierte Programmierung mit Python. Bonn (MITP) 2004.

[We05] Weigend, Michael: Python-Gepackt. Bonn (MITP) 2. Aufl. 2005.