Mit Klebezettel und Augenbinde durch die Objektwelt

Ira Diethelm[1,2], Leif Geiger[2], Albert Zündorf[2]

[1]Gaußschule
Löwenwall 18a
D-38100 Braunschweig

[2]SE, Universität Kassel
Wilhelmshöher Allee 73
D-34121 Kassel

(ira.diethelm | leif.geiger | albert.zuendorf)@uni-kassel.de

http://www.se.eecs.uni-kassel.de/se/

Abstract: Eine der wesentlichen Schwierigkeiten im Anfangsunterricht für Informatik ist der für das Schreiben und Verstehen von Programmen notwendige Perspektivwechsel. Dies gilt insbesondere bei der Verwendung objektorientierter Datenstrukturen. Die Aufgabenstellung gibt den Schülern typischerweise erst einmal eine Gesamtsicht auf das Problem, zum Beispiel bei der Problemanalyse mit Objektdiagrammen. Der Computer oder das Programm hat aber während der Programmabarbeitung zu jedem Zeitpunkt immer nur lokale Informationen in Form von lokalen Variablen, Parametern oder Objektattributen zur Verfügung. Aufgrund seiner Gesamtsicht auf das Problem findet der Schüler oft sehr schnell eine Problemlösung oder ein Problemlösungsverfahren. Bei der Umsetzung solch einer Strategie in ein Programm stellt er dann aber fest, dass einzelne für ihn leichte Lösungsschritte mit den lokalen Informationen eines Programms nicht ohne weiteres zu lösen sind. Zur Erleichterung des notwendigen Perspektivwechsels insbesondere in der objektorientierten Programmierung stellt dieses Papier zwei didaktische Hilfsmittel vor: Augenbinden und Klebezettel.

1 Einleitung

Im Informatikunterricht tritt häufig der Fall auf, dass die Schüler den Unterrichtsgegenstand aus einer anderen Perspektive betrachten müssen, um Lösungsstrategien oder ein Modell zu entwickeln. Dies gilt insbesondere für objektorientierten Unterricht, wenn die Problemstellung mit Hilfe von Objektdiagrammen analysiert wird. Bei der Diskussion der Objektdiagramme scheint das Problem verstanden, bei der Umsetzung in ein Programm haben die Schüler aber plötzlich große Schwierigkeiten. Es fällt ihnen schwer, sich in die Rolle des Computers zu versetzen und sich vorzustellen, welche Möglichkeiten dieser hat und welchen Einschränkungen er unterliegt und in welchen Programmschritten sie das Problem lösen können. Dies ist meist genau der Punkt, den man in der Unterrichtsstunde oder in der Einheit als Hauptschwierigkeit identifizieren würde. Die Schüler können an diesem Punkt mit ihrer allwissenden Draufsicht auf das analysierte System nur schwer weiterkommen.

Insbesondere im Anfangsunterricht ist an solchen Stellen der Lehrer gefragt, Hilfen für die Schüler anzubieten, um diesen Perspektivwechsel zu begünstigen oder gar zu erzwingen. Es gibt zwei Unterrichtssituationen, bei denen dieses Problem auftritt:

1. bei der Analyse und Dekonstruktion eines bestehenden Systems.
2. bei der Konstruktion eines neuen Systems als Lösung für ein Problem.

In beiden Fällen muss sich der Schüler in einer Erkundungsphase in die Lage des Computers versetzen und sich dessen beschränkte Möglichkeiten bewusst machen: „Mit welchen Werten, anderen Objekten, Methoden kann ich hier Anderungen durchführen?“, „Wie finde ich benötigte Informationen / Partnerobjekte?“. Ebenso muss der Schüler später in einer Testphase die Arbeitsweise des Programms nachempfinden können, was man bei prozeduralen Ansätzen z. B. mit Tracetabellen versucht.

Für diese Fälle bieten wir in diesem Artikel anhand eines Beispiels Hilfen an, die im objektorientierten Unterricht ohne großen Aufwand sehr effektiv genutzt werden können: Für die Erkundungsphase benötigen wir lediglich eine Augenbinde und beschreiben dies in Kapitel 2 und für die Testphase benutzen wir Klebezettel in Kapitel 3. Eine Ubertragung dieser Hilfen auf den prozeduralen Unterricht ist durchaus denkbar.

2 Augenbinde

Betrachten wir das folgende Beispiel: Die Schüler sollen für das bekannte Spiel „Mensch ärgere dich nicht“ ein Programm erstellen, mit dem man das Spiel spielen oder die Arbeitsweise eines solchen Programms erforschen kann. Ein zentraler Punkt bei diesem Spiel ist das Weitersetzen des Spielsteins ohne Rauswerfen, nachdem man eine Zahl zwischen eins und fünf gewürfelt hat, z. B. eine drei.

Um eine Implementierung für die obige Spielsituation zu finden, betrachten wir zuerst einen Beispielablauf. Hierfür beginnen wir mit der Modellierung der Ausgangssituation: Ein paar Felder befinden sich nebeneinander in einer Reihe, ein Spielstein befindet sich auf einem dieser Felder, der Würfel zeigt eine 3 und auf den drei nächsten Feldern in Spielrichtung befinden sich keine weiteren Spielsteine. Der Spielstein und der Würfel gehören einem Spieler. Abbildung 1 zeigt ein UML Objektdiagramm, das diese Situation modelliert.

Wenn die Schüler auf dieses Objektdiagramm schauen, glauben sie aufgrund ihrer Gesamtsicht meist sofort zu wissen, wie der Endzustand erreicht wird: Die Verbindung zwischen dem Spielstein und Feld *fb* muss zerstört und eine neue zwischen dem Spielstein und Feld *fe* muss erzeugt werden. Ebenfalls haben sie aufgrund der Draufsicht das Gefühl, sie wüssten, wie dies innerhalb eines Programms für den allgemeinen Fall geschieht. Sollen sie dann aber dieses Programm schreiben, stellen sie fest, dass einem Spielstein-Objekt zunächst nur das aktuelle Feld bekannt ist und dass die Operation „finde Zielfeld“ oder „finde Feld fe“ in ihrer Programmiersprache nicht direkt angeboten wird. Unglücklicherweise tritt dieses Mismatch-Problem häufig gerade dann auf, wenn die Schüler allein in

einer Hausaufgabe oder am Terminal die ersten selbständigen Programmierschritte machen. Dies führt dann zu großer Frustation, eine unnötige Lernhürde entsteht.

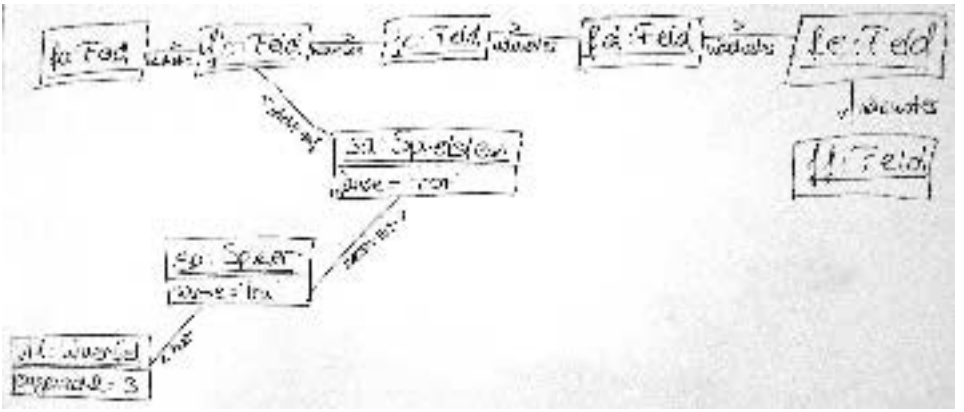

Abbildung 1: Objektdiagramm für die Ausgangsituation

Um dieses Problem zu vermeiden, sollten die Schüler den notwendigen Perspektivwechsel besser schon während der Gruppenarbeit beim Entwurf der Objektdiagramme vollziehen. Dies ist auch für eine konkretere Diskussion von Designentscheidungen bei der Erstellung der Objektdiagramme eine wichtige Voraussetzung. Ohne diesen Perspektivwechsel können Fragen wie „Brauchen wir da nicht noch einen Link?“ oder „In welchem Attribut merken wir uns das am Besten?“ nicht sinnvoll entschieden werden.

Um die Schüler die eingeschränkten Möglichkeiten von Objekten erfahren zu lassen, bilden wir das gezeigte Objektdiagramm in einem Rollenspiel nach. Jedem Schüler werden die Augen verbunden und ein Schüler, der kein Objekt spielt, stellt die Links zwischen den Objekten her, indem er die Hände der Objekt-Schüler so miteinander zusammenbringt, dass das Objektdiagramm genau abgebildet wird. Um die Richtung eines Links zu signalisieren, können auch nur die Arme des Schülers als Link dienen, von dem die Verbindung ausgeht, wie in Abbildung 2.

Der Schüler, der den Spielstein repräsentiert, soll nun die Methode *setzen()* ausführen. Er wird dadurch zum zentralen Handelnden. Der Spielstein-Schüler erkennt als erstes, dass das Spielstein-Objekt noch gar nicht weiß, wie weit er vorgehen soll und ist gezwungen, dies über den Spieler-Schüler vom Würfel-Schüler zu erfragen und sich anschließend irgendwo zu merken. In unserem Beispiel merkt sich der Spielstein-Schüler die Augenzahl (hier 3) mithilfe der Finger der freien Hand. Alternativ kann ein weiterer Schüler die Rolle einer Variablen übernehmen, er braucht sich die Augen nicht zu verbinden und notiert sich den aktuellen Variablenwert auf einem Zettel oder an der Tafel. Dass die Augenzahl im allgemeinen erst noch erfragt werden muss, wird noch deutlicher, wenn die Augenzahl vom Würfel erst „erwürfelt“ wird, nachdem sich die Schüler die Augen verbunden haben.

Abbildung 2: Objektdiagramm mit Schülern

Anschließend muss der Spielstein-Schüler den Feld-Schüler, mit dem er verbunden ist, fragen, wer dessen Nachfolger ist und seinen „steht auf“ Arm zu diesem Nachfolger vorrücken. Nun muss sich der Spielstein-Schüler merken, dass er nur noch zwei Felder gehen muss und beginnt wieder nach dem Nachfolgefeld zu fragen usw., bis er nicht mehr gehen muss.

Wie er handelt, teilt der Schüler den anderen bislang unbeteiligten Schülern mit, die dies an der Tafel dokumentieren. Sie erstellen damit eine sehr genaue textuelle Beschreibung des Ablaufs der Methode, zunächst nur für die Augenzahl 3, später auch für andere Augenzahlen. Der Vergleich dieser Beschreibungen liefert eine gute Hilfe für die Implementierung der Methode und für das Verständnis ihrer Arbeitsweise. Abbildung 3 zeigt den Ablauf der *setzen()* Methode im „Objektspiel mit Augenbinde“.

Natürlich muss man den Schülern nicht die Augen verbinden, es reicht theoretisch völlig aus, die Schüler anzuweisen, die Augen zu schließen. Dennoch hat die Augenbinde für die Schüler, denen es nicht zu unangenehm ist, eine einschränkende Wirkung, die durch das bloße Augenschließen nicht erreicht werden kann: Sie ist nicht freiwillig. Selbst, wenn man die Augen öffnet, kann man die Umgebung nicht mehr wahrnehmen als zuvor. Man kann die Gesamtsituation nicht mehr einfach „überblicken“, sondern man ist auf das Befragen von Nachbarn und auf das Merken einfacher Informationen reduziert. Dies macht den Schülern in etwa erfahrbar, welchen Einschränkungen ein Objekt in einem objektorientierten Programm unterliegt. Anstelle einer Augenbinde kann dieser Effekt auch mit einer tief ins Gesicht gezogenen Kappe oder mit einem als Blende benutzten Klebezettel erreicht werden.

Abbildung 3: Sequenz mit Schülern

3 Klebezettel

Bei der Erarbeitung der Funktionsweise fremder gegebener Programme, wird traditionell in der imperativen Programmierung eine Tracetabelle genutzt. Für die objektorientierte Programmierung treten hier spätestens dann Probleme auf, wenn die Objektstruktur verändert wird, z. B. wenn ein neuer Link erzeugt wird. Die Veränderung der Objektstruktur kann mit solchen Tabellen nicht protokolliert werden. Möglicherweise ist dies auch einer der Gründe, wieso Objektorientierung als so schwer und für Anfänger ungeeignet angesehen wird.

Um diese Schwierigkeit aus der Welt zu räumen, schlagen wir vor, die Tracetabelle gegen Klebezettel (z. B. Post-its) und eine Kamera einzutauschen. Die Grundstruktur dieses Vorgehens besteht aus folgenden Schritten:

1. Die Ausgangssituation wird in Form eines Objektdiagramms modelliert.
2. Das Programm wird Anweisung für Anweisung durchgegangen.
3. Wird bei der schrittweisen Ausführung des Programms ein Objekt in einer lokalen Variablen zwischengespeichert, so wird auf das Objekt ein Zettel mit dem Namen der Variablen geklebt. Aus Sicht des Programms wird das Objekt ab jetzt mit diesem neuen Namen angesprochen.
4. Für einfache z. B. zahlwertige Variablen werden zusätzliche Trace-Eintragungen auf der Tafel gemacht.
5. Modifikationen der Objektstruktur werden an der Tafel durch Auswischen und Neuzeichnen nachvollzogen.
6. Jeder Schritt wird fotografiert und die Abfolge der Fotos ergibt als eine Art Comicstrip den Verlauf der Methode.

Im Folgenden soll dieses Vorgehen an dem genannten Beispiel erläutert werden. Es soll die Funktion der Methode *setzen()* analysiert werden. Diese Methode setzt den Spielstein jeweils ein Feld nach vorn und zwar so oft, wie es die Augenzahl des Würfels angibt. Nachfolgend ist der Java-Quelltext der Methode aufgeführt:

```
1   public class Spielfigur { ...
2      public void setzen() {
3         Spieler meinSp = this.getSpieler () ;
4         Wuerfel meinW = meinSp.getWuerfel() ;
5         int nochGehen = meinW.getAugenzahl() ;
6
7         while ( nochGehen>0 ){
8            Feld aktF = this.getFeld () ;
9            Feld naechstesF = aktF.getNachfolger ();
10           this.setFeld (null);
```

```
11            this.setFeld (naechstesF);
12            nochGehen = nochGehen-1;
13  }   }   }
```

Alternativ kann man die Methode auch mit Fujabas *Regeldiagrammen* modellieren. Abbildung 4 zeigt ein solches Diagramm, das dasselbe Verhalten modelliert, wie obiger Quelltext. In [DGZ02] werden Regeldiagramme näher erläutert.

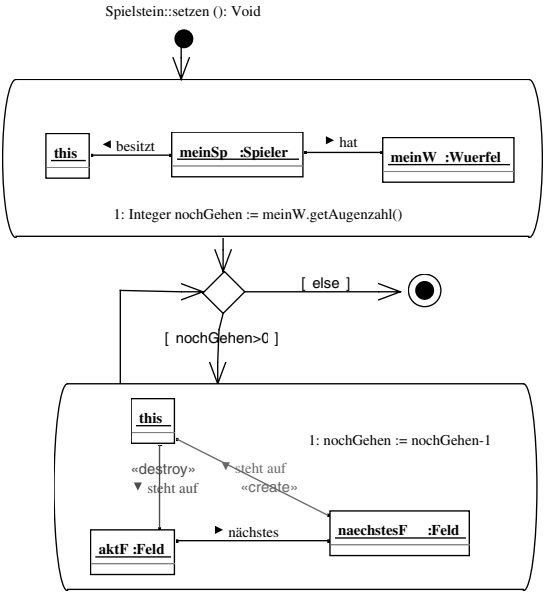

Abbildung 4: Regeldiagramm für setzen()

Um nun die Funktion der Methode zu verstehen, analysieren wir ihre Wirkung auf eine bestimmte Objektstruktur. Als Anfangssituation soll hier wieder das Objektdiagramm aus Abbildung 1 dienen. Wendet man die Methode *setzen()* auf das Spielstein-Objekt *sa* im Objektdiagramm aus Abbildung 1 an, so werden in Zeile 3, 4 und 5 bzw. in der ersten Aktivit ät des Regeldiagramms die Objekte für den Spieler und den Würfel identifiziert und mit *meinSp* bzw. *meinW* benannt. Hierbei wird als erstes *sa*, das Objekt, auf dem die Methode aufgerufen wurde, als this bezeichnet. Anschließend wird das Objekt *sp* als *meinSp* und *w1* als *meinW* benannt. Hierfür kleben wir Klebezettel mit den Beschriftungen *this*, *meinSp* und *meinW* auf die entsprechenden Objekte. Es bietet sich an, die Objektnamen im Diagramm zu überkleben und gleichzeitig darauf zu achten, dass der Klassenname noch lesbar ist. Außerdem wird eine lokale Variable vom Typ Integer mit Namen nochGehen angelegt und mit dem Wert der Augenzahl von *meinW* belegt. Hierfür kleben wir einen Klebezettel mit der Beschriftung *nochGehen* an einen beliebigen Platz im Objektdiagramm und schreiben mit einem Stift den Wert hinzu. Das erste Bild von Abbildung 4 zeigt das Ergebnis dieser Phase.

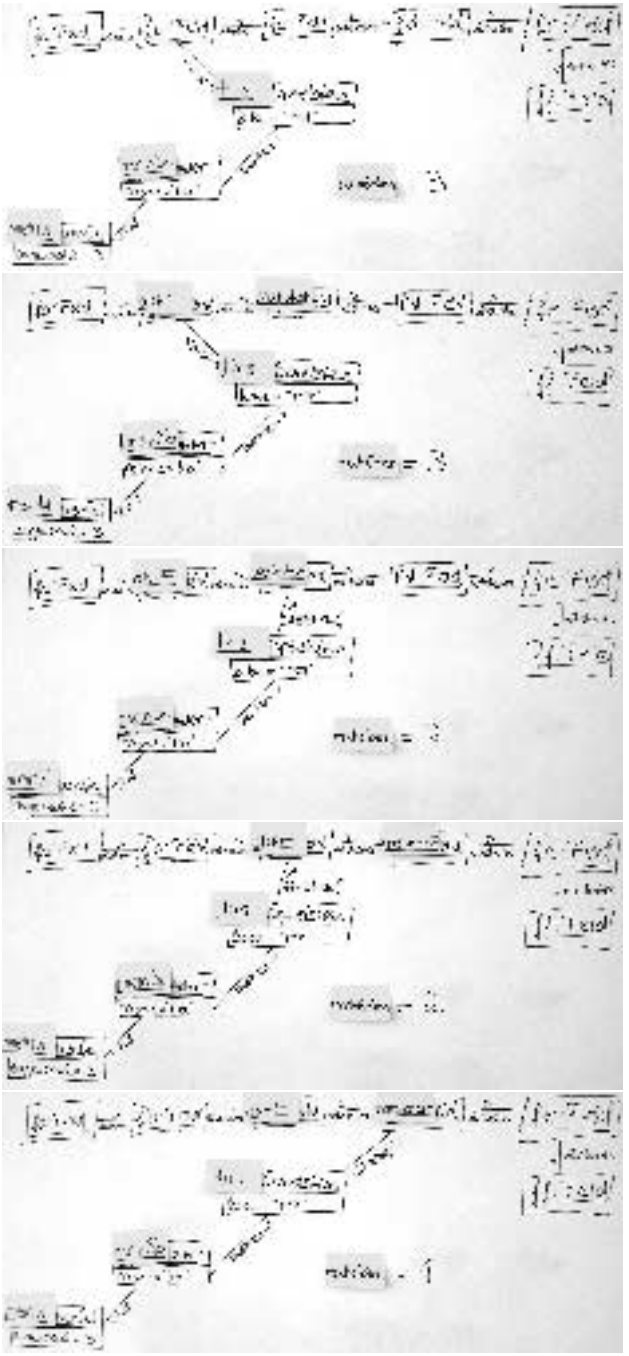

Abbildung 5: Tafelbilder mit Postits als Trace

Zu Beginn der Schleife in Zeile 8 und 9 bzw. in der zweiten Aktivität von Abbildung 4 werden die Feld-Objekte für das aktuelle Feld und das nächste Feld identifiziert: Das Feld *fb* wird so zu *aktF* und *fc*zu *naechstesF*. Sie werden mit entsprechenden Klebezetteln beklebt, vgl. zweites Bild von Abbildung 3. In Zeile 11 und 12 wird der steht auf Link von *aktF* auf *naechstesF* geändert. Daher wischen wir hier den Link auf der Tafel weg und ziehen einen entsprechenden neuen. In Zeile 13 wird noch *derWert* von *nochGehen* um eins vermindert. Dies wird ebenfalls auf der Tafel geändert. Am Ende des ersten Schleifendurchlaufs erhält man somit das 4. Tafelbild.

Während des zweiten Schleifendurchlaufs wird wieder das aktuelle und das nächste Feld zugeordnet. Nun wird *fc* zu *aktF* und *fd* zu *naechstesF*, daher müssen wir hier die Klebezettel umkleben, vgl. 5. Bild. Nun wird wieder in Zeile 11-13 der steht auf Link zerstört und zu *naechstesF* ein neuer gezogen und *nochGehen* auf 1 gesetzt, vgl. Bild 6.

Nach einem weiteren Schleifendurchlauf, in dem man wieder die Klebezettel *aktF* und *naechstesF* um ein Feld versetzt, wieder den Link ändert und *nochGehen* denWert 0 erhält, sieht man das Tafelbild aus Abbildung 6.

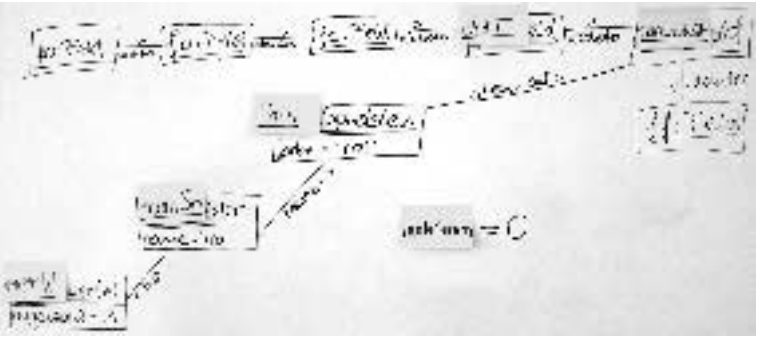

Abbildung 6: nach dem Ende der Schleife

Bei der schrittweisen Ausführung des Programms empfiehlt es sich, die fertigen Schritte jeweils abzuhaken. In Schleifen kann auch ein kleiner Pfeil als eine Art Program-Counter verwendet werden, der immer wieder weggewischt und neu gezeichnet wird.

Wird ein Methodenaufruf abgearbeitet und man betritt den Rumpf der neuen Methode, so wechselt man auch den Kontext an lokalen Variablen. Das heißt, die alten Klebezettel an der Tafel sind bis zum Methodenrücksprung nicht mehr gültig. Da es zu mühselig wäre, alle alten Klebezettel zu entfernen und sie beim Rücksprung in die alte Methode richtig wieder aufzukleben, empfehlen wir, die alten Klebezettel einfach hängen zu lassen und für die neue Methode Klebezettel in einer anderen Farbe oder Form zu verwenden. Für die typischerweise kleinen Unterrichtsbeispiele kommt man meist mit wenigen Farben und Formen aus. Ein Musterzettel der verwendeten Farbe oder Form wird auch beim jeweiligenMethodenrumpf hingeklebt, um die Zuordnung zwischen lokalen Objektnamen und Methoden eindeutig zu machen. Beim Methodenrücksprung werden die Klebezettel, die zu diesem Methodenaufruf gehören, von der Tafel genommen. Die Klebezettel, die zu der jetzt wieder aktiven Methode gehören, sind damit die aktuell gültigen. Bei rekursiven

Methodenaufrufen werden die verschiedenfarbigen Klebezettel für jede Methodeninkarnation bei den Objekten und bei dem Methodenrumpf leicht versetzt übereinander geklebt. Dadurch entsteht ein Stapel von Klebezetteln, der im Prinzip das Prozedurkellerverhalten veranschaulicht.

Vor der Verwendung der Klebezettel haben wir bei der schrittweisen Programmausführung zusätzlich zu dem aktuellen Objektdiagramm eine Wertetabelle für lokale Variablen und Parameter verwendet. Bei zeigerwertigen Variablen wurden die Objektbezeichnungen als Werte in die Tabelle eingetragen. Eine solcheWertetabelle ermöglicht zwar im Prinzip auch eine schrittweise Programmausführung, sie stellt aber schlicht eine zusätzliche Indirektionsstufe dar, die das Verständnis erschwert. Durch die Einführung der Klebezettel wurde die Programmausführung für die Schüler viel plastischer und leichter nachvollziehbar.Wir haben den Eindruck, dass durch die Klebezettel bei den Schülern auch ein viel einfacheres mentales Modell für lokale Variablen entsteht. Lokale Variablen sind einfach temporäre (Spitz-)Namen für verwendete Objekte. Diesen so benannten Objekten kann man dann im Programm Kommandos geben.

4 Zusammenfassung

Wir verwenden Augenbinden und Klebezettel jetzt schon seit einigen Jahren mit großem Erfolg im Anfangsunterricht der Sekundarstufe II in der Gaußschule Braunschweig und zum Teil auch in Einführungsvorlesungen zur Objektorientierung an der Universität Kassel. Beide Hilfsmittel haben sich für die Uberwindung zweier schwieriger Lernhürden hervorragend bewährt. Durch die Rollenspiele mit Augenbinden erhalten die Schüler und Studenten eine sehr gute Vorstellung von den Möglichkeiten und vor allem von den Beschr änkungen eines Objekts in einem objektorientierten Programm. Es wird ihnen viel klarer, bis auf welche Detailstufe sie ein Problem runterbrechen müssen, um auf die Ebene von Programmanweisungen zu kommen.

Wir glauben, dass sich durch die Rollenspiele mit den Augenbinden bei den Schülern eine mentale Vorstellung von der Arbeitsweise von Objekten bildet, die ihnen bei der späteren Problemlösung und Programmierung sehr hilft.

Weiterhin glauben wir, dass die Schüler eine geeignete mentale Vorstellung von der Arbeitsweise eines Computers beziehungsweise eines objektorientierten Programms benötigen, um Programme verstehen und schreiben zu können. Für Programme mit Zeigerstrukturen (verkettete Listen, Bäume, etc.) und für Programme mit umfangreichen Objektstrukturen ist dabei eine explizite Darstellung der verwendeten Daten sehr hilfreich. Dies wird in unserem Ansatz durch die Objektdiagramme erreicht. Eine weitere Lernhürde sind erfahrungsgemäß lokale Variablen und das Prozedurkeller-Verhalten bei Rekursion. Hier verhelfen unsere Klebezettel den Schülern zu einer einfachen, aber tragfähigen Vorstellung.

Literaturverzeichnis

[Ba98] Helmut Balzert: Lehrbuch der Software - Technik 1, 2. Aufl., Spektrum, 1998.

[DGZ02] I. Diethelm, L. Geiger, A. Zündorf: UML im Unterricht: Systematische objektorientierte Problemlösung mit Hilfe von Szenarien am Beispiel der Türme von Hanoi. in Forschungsbeiträge zur „Didaktik der Informatik“ - Theorie, Praxis und Evaluation; GI-Lecture Notes, pp. 33-42 (2002)

[Di02] I. Diethelm, L. Geiger, T. Maier, A. Zündorf: Turning Collaboration Diagram Strips into Storycharts;Workshop on Scenarios and state machines: models, algorithms, and tools; ICSE 2002, Orlando, Florida, USA, 2002.

[Fu02] Fujaba Homepage, Universität Paderborn, http://www.fujaba.de/.

[Hu00] P. Hubwieser: Didaktik der Informatik - Grundlagen, Konzepte, Beispiele, Springer Verlag, Berlin, 2000.

[Kö00] H. Köhler, U. Nickel, J. Niere, A. Zündorf: Integrating UML Diagrams for Production Control Systems; in Proc. of ICSE 2000 - The 22nd International Conference on Software Engineering, June 4-11th, Limerick, Ireland, acm press, pp. 241-251 (2000)

[li02] life3-Homepage, Universität Paderborn, http://life.uni-paderborn.de/.

[SN02] C. Schulte, J. Niere: Thinking in Object Structures: Teaching Modelling in Secondary Schools; in Sixth Workshop on Pedagogies and Tools for Learning Object Oriented Concepts, ECOOP, Malaga, Spanien, 2002.

[Zü01] A. Zündorf: Rigorous Object Oriented Software Development, Habilitation Thesis, University of Paderborn, 2001.