Dramatisieren und literarisches Programmieren

Michael Weigend

Institut für Didaktik der Mathematik und der Informatik
Westfälische Wilhelms-Universität Münster
Fliednerstr. 21
48149 Münster
michael.weigend@uni-muenster.de

Abstract: In einer Studie mit 119 Teilnehmern wurde untersucht, inwieweit Schülerinnen und Schüler im Alter von 14 bis 18 ohne spezielle Informatik-Vorkenntnisse metaphorisch formulierte Algorithmen nachvollziehen können und selbst Metaphern zur Verschriftlichung eigener Anleitungen verwenden. Die Kompetenz, eine ansprechende und verständliche literarische Form für einen Algorithmus zu finden, ist eine Kompetenz, die zur Allgemeinbildung gehört und für informatisches Modellieren nützlich ist.

1 Algorithmus und Drama

Ein Computerprogramm kann man sich als abstrakte Form eines Dramas vorstellen. Da werden Akteure beschrieben und Operationen, die sie in bestimmten Situationen ausführen. Freilich verläuft jede Ausführung etwas anders. Es ist wie beim Improvisationstheater. Vorgegeben ist ein allgemeines Muster, etwa eine Rollenaufteilung. Die konkrete Ausführung hängt in der Regel auch von unvorhersehbaren Ereignissen oder Daten ab, die aus der Umgebung stammen. Eine Reihe von Autoren, allen voran Donald Knuth, sehen Computerprogramme als Literatur. „I believe that the time is ripe for significantly better documentation of programs, and that we can best achieve this by considering programs to be works of literature." [Kn84, S. 99]. Schließlich ist ein Programm ein Text, der zwar auch von Maschinen interpretiert werden kann, der aber im Wesentlichen für Menschen geschrieben ist und von Menschen verstanden werden muss. Knuth prägte für diese Sichtweise den Begriff „literarisches Programmieren" (literate programming). Claudia Herbst [He02] kritisiert, dass Programme nur von wenigen (meist männlichen) Eingeweihten produziert und verstanden werden. „Among code's most defining characteristics are its inaccessibility and covert nature." [He02, S.3]. Zumindest im Open Source-Bereich trifft diese Behauptung nicht zu. Offener Programmtext wird veröffentlicht und muss sich auf einem „Literaturmarkt" eigener Art behaupten. Beispiel: Im Python Package Index, einem öffentlichen Repository für Python Software, waren am 28. Januar 2011 13092 Python Module gespeichert (http://pypi.python.org/pypi). In den Beschreibungen der Module werden sogar Zielgruppen (intended audience) angegeben (Juristen, Wissenschaftler etc.). Wenn Open Source-Programmtexte in der Konkurrenz erfolgreich sein sollen, müssen sie gut formuliert sein. Denn der Leser (Programmierer) hat die Wahl und wird das Produkt nehmen, das er oder sie am besten versteht. Was für den

Quelltext einer Software gilt, gilt erst recht für die Dokumentation. Sie soll im Großen die Struktur und Arbeitsweise einer kompletten Software erklären (Projektmetapher) und im Kleinen (meist in Form von Kommentaren) algorithmische Ideen einzelner Funktionen vermitteln.

1.1 Was ist ein dramatisierter Algorithmus?

Aristoteles [Ar69] beschreibt in seinen Ausführungen zur Tragödie sechs Bestandteile eines Dramas. Dazu gehören die *Handlung*, die ein in sich geschlossenes Ganzes bildet und *Charaktere*, die die Handlung ausführen. Die Charaktere sind so gewählt, dass die Handlung nicht willkürlich erscheint sondern „schicksalhaft“ und als logische Folge festgelegter Charaktermerkmale und Verhaltensdispositionen der Akteure.

In einem dramatisierten Algorithmus sind die Akteure und Handlungen Metaphern für abstrakte algorithmische Elemente. In der Rhetorik versteht man unter einer Metapher die Übertragung eines Inhalts aus einem Wissensbereich (Quelldomäne) auf ein völlig anderes Gebiet (Zieldomäne). Man verwendet Metaphern um einen Text sprachlich interessanter und abwechslungsreicher zu machen [Ba05].

Wenn die Metapher aus einer dem Leser vertrauten Domäne stammt, kann sie auch zum Verständnis beitragen. Der Rezipient kann dann sein bereits vorhandenes Wissen auf eine neue Domäne anwenden. Eine LIFO-Datenstruktur (last in first out) wird metaphorisch als Schlange bezeichnet. Man stellt sich eine Warteschlange an einer Supermarktkasse vor und kann dieses Wissen direkt verwenden um die Funktionalität der Schlange zu verstehen.

Nun kann man auch Aktivität abstrakter geometrischer Gebilde beschreiben: „Die Kreisfläche bewegt sich nach links.“ Wozu also Metaphern? Folgt man der aristotelischen Idee des Dramas, muss die Aktivität in der Beschaffenheit des Akteurs ihren Ursprung haben. Einer Kreisfläche sieht man es nicht an, dass sie ihre Position verändern kann. Bezeichnet man sie metaphorisch als Schildkröte (also als bewegliches Lebewesen), so impliziert dies bereits die Möglichkeit einer Ortsveränderung.

Ein dramatisierter Algorithmus enthält oft auch dekorative Elemente, die das Szenario auskleiden aber kein Pendant im Algorithmus haben. Man könnte sie also weglassen ohne die algorithmische Struktur zu verfälschen. Manchmal sind hinzuerfundene Elemente aber wichtig, um Einzelteile zu einer geschlossenen Gestalt (einer plausiblen Handlung) abzurunden. Abbildung 1 stammt aus einer Aufgabe, die in der Studie verwendet wurde, die später in diesem Beitrag vorgestellt wird.

Das Bild zeigt eine Anordnung aus Formen, die so verändert werden soll, dass Buchstaben entstehen (hier: TEL).

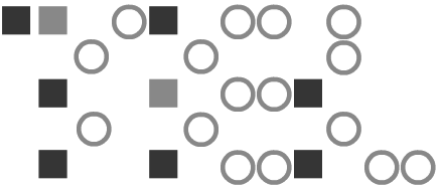

Abbildung 1: Eine Anordnung von Formen, die durch einen Algorithmus verändert wird.

Diese Veränderung wird durch einen einfachen Algorithmus beschrieben:

Am Strand liegen eckige Steine und runde Schildkröten. Jemand ruft „Schildkröten, nach links!“ Und alle Schildkröten gehen ein Stück nach links.

Dies ist ein dramatisierter Algorithmus. Er enthält Metaphern und eine (unscheinbare) Dekoration. Offenbar sind die Steine und Schildkröten Metaphern für Quadrate und Kreise. Der Begriff „Strand“, der Ort des Geschehens, ist für den *Algorithmus* unwichtig. Für das *Drama* ist er jedoch durchaus relevant, weil er die Handlung noch weiter konkretisiert. Man könnte noch weitere Elemente erfinden, etwa eine Ursache für das Geschehen. Eine nicht-dramatisierte Version lautet: *Schiebe alle Kreise ein Stück nach links*.

1.2 Dramatisierung und Software-Entwicklung

In jedem Styleguide findet man den Grundsatz, dass man sinnvolle („sprechende“) Bezeichner wählen soll. Die Bezeichnerwahl geht häufig mit einer Dramatisierung einher, die quasi neben der rein technischen Struktur- und Algorithmus-orientierten Programmentwicklung steht. Wenn eine Variable *summe* deklariert wird, ist allein in dieser Benennung eine Rolle (also auch Verhaltensmuster) definiert: Die Variable wird die *Summe* irgendwelcher Einzelwerte speichern (und nicht etwa ihr Produkt).

Sajaniemi [Sa02] hat die Rollen untersucht, die Variablen in Programmbeispielen in Informatik-Lehrbüchern spielen. Er beschreibt allgemeine Muster für Rollen wie z.B. *Konstanten*, die sich nicht ändern, oder *Stepper*, deren Inhalt in Iterationen um einen konstanten Betrag erhöht oder erniedrigt werden.

Im traditionellen Software-Engineering wird spätestens in der Entwurfsphase eine in sich schlüssige kohärente Welt, ein informatisches Modell (z.B. als Klassenstruktur) entwickelt, die als Grundlage für die Namenswahl für die einzelnen Elemente dient. Bei einer agilen Softwareware-Entwicklung (z.B. Extreme Programming [B299]), entsteht der dramaturgische Kern des Projektes unter Umständen erst während des Prozesses. Man beginnt mit einer eher vagen „Projektmetapher“, die in einer einzigen Gestalt die Idee der Systemstruktur zum Ausdruck bringt. Erst im Verlauf des Projekts wird sie in Iterationen immer weiter entfaltet. Dabei kommt es vor, dass (im Rahmen eines Refactoring) Bezeichner des bereits funktionierenden Programms noch geändert werden, um die Begrifflichkeit konsistent zu halten. Dabei wird allein die literarische Qualität und nicht die algorithmischen Effizienz verbessert. Die Idee des Dramas als Interaktion

von Entitäten ist in der Objektorientierten Programmierung (OOP) kultiviert und formalisiert. Wer ein objektorientiertes Programm konzipiert, muss eine animierte Fantasiewelt eigenaktiver Entitäten erfinden.

- In einem objektorientierten Grafikprogramm wird eine Linie nicht mit Hilfe eines Stiftes verlängert sondern sie erhält den Auftrag, sich selbst länger zu machen. Sie ist (im Gegensatz zum realen Bleistiftstrich) eigenaktiv und wächst.

- In objektorientierten GUI-Implementierungen wird ein flüchtiges Ereignis wie z.B. ein Mausklick , zu einer dauerhaften Entität materialisiert, die von einem Eventhandler verarbeitet werden kann.

Die Beispiele zeigen, dass die Objekte eines Programms durch die beobachtbare Wirklichkeit zwar inspiriert, aber nicht von ihr einfach abgeleitet werden können. Programmieren ist Fiktion und nicht Dokumentation. Da die Objekte interagieren, gibt es eine Rollenverteilung und induzierte Handlungsabläufe wie in der griechischen Tragödie. Die Programmiersprache erlaubt, einer Klasse praktisch beliebige Operationen zuzuordnen. Die Kunst liegt darin, *sinnvolle* Konglomerate von Aktivität und Daten zu erschaffen. Ziel ist eine Entlastung des Arbeitsgedächtnis [De08]. Die ist dann gegeben, wenn eine Klasse vertraute Ganzheiten, – Gestalten im Sinne der Gestaltpsychologie [We25] repräsentiert. Dann kann man auch ohne Detailkenntnisse der Klassendefinition erahnen, welche Attribute und Methoden Objekte einer Klasse besitzen und welche nicht, und für welche Zwecke man sie verwenden kann.

2 Das Design der Studie

Dramatisieren ist eine Facette des Programmierens. Wer ein Computerprogramm entwickelt, braucht die Fähigkeit kleine, schlüssige Geschichten (genauer: Muster für Geschichten) zu erfinden, die Problemlösungen repräsentieren. Dazu gehört sprachliche Fantasie. Denn es müssen Metaphern und Dekorationen erfunden werden. Im Unterschied zu naturwissenschaftlichen Modellen, die ein strukturtreues Abbild eines Wirklichkeitsausschnittes liefern, sind informatische Modelle in ihrer Struktur durch die Wirklichkeit bestenfalls angeregt. Das wichtigste Qualitätsmerkmal eines informatischen Modells ist kognitive Beherrschbarkeit. Für den Informatikunterricht ergeben sich einige Fragen: Wie gut können Schülerinnen und Schüler metaphorisch formulierte Algorithmen verstehen? Können sie sie besser oder schlechter nachvollziehen als sachlich formulierte umgangssprachliche Algorithmen? Inwieweit kann man Schülerinnen und Schüler durch Metaphern (in Wort und Bild) inspirieren, eigenständig einen Algorithmus zu formulieren? Werden bestimmte Metaphern gegenüber anderen bevorzugt?

In den Jahren 2010 und 2011 habe ich mit insgesamt 119 Schülerinnen und Schülern aus den Jahrgangsstufen 9, 10 und 11 einer Gesamtschule Workshops zur naiven Algorithmik durchgeführt. Die Schüler hatten keine besonderen

Programmiervorkenntnisse. Nur einige wenige hatten Informatikunterricht, der sich aber nur sporadisch mit Programmierung beschäftigte.

Es gab zwei etwas unterschiedliche Typen von Workshops (W1 und W2) mit folgenden Teilnehmerzahlen:

W1: 58 Schüler, davon 28 m., 30 w., Durchschnittsalter 15,4 Jahre ($\sigma = 0{,}6$)
W2: 61 Schüler, davon 21 m., 40 w., Durchschnittsalter 15,4 Jahre ($\sigma = 1{,}1$)

Sie dauerten jeweils etwa 45 min und gliederten sich in drei Phasen: 1) Interpretation von vorgegebenen Algorithmen. 2) Entwicklung eigener Algorithmen 3) Test der eigenen Algorithmen. Ich beginne mit der Darstellung des ersten Typs W1 und beschreibe dann, anschließend, was beim zweiten Typ anders war.

In Phase 1 erhielten die Schüler ein Aufgabenblatt das zwei dramatisierte und zwei nicht-dramatisierte Algorithmen mit Bildern wie in Abbildung 1 enthält. Die Teilnehmer führten die Algorithmen jeweils aus, suchten das Lösungswort (eine sinnlose Buchstabenkombination) und schrieben es auf. Zum Schluss markierten sie die schwierigste und die leichteste Aufgabe.

In Phase 2 sollten die Teilnehmer selbst Algorithmen mit Metaphern entwickeln und konnten sich dazu aus einem Reservoir von insgesamt zehn Aufgabenblättern bedienen.

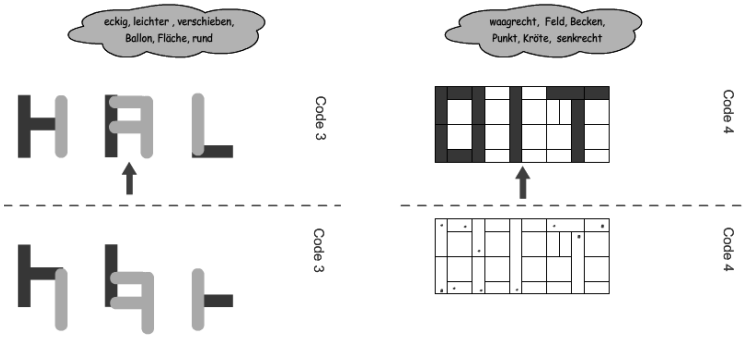

Abbildung 2: Eine Anordnung von Formen, die durch einen Algorithmus so verändert wird, dass drei Buchstaben entstehen.

Abbildung 2 zeigt zwei Beispiele. Auf dem Blatt sieht man zwei Bilder. Der Algorithmus beschreibt, durch welche Veränderungen man das untere Bild in das obere verwandeln kann. Dabei sollen wieder (wie in Phase 1) Buchstabenkombinationen entstehen. Die Probleme sind so aufgebaut, dass sie durch einen Algorithmus folgenden Typs gelöst werden: Eine einfache Aktivität (z.B. etwas verschieben oder eine Fläche ausfüllen) wird auf eine bestimmte Auswahl von grafischen Elementen angewendet. Es müssen also Kollektionen aufgrund gemeinsamer Merkmale definiert und Iterationen über diese Kollektionen ausgeführt werden. Die Probleme waren so gestellt, dass eine

Formulierung ohne Verwendung einer Iteration der Art „Mache mit allen ... folgendes...“ lang und komplex würde.

Die Teilnehmer der Workshops mussten also zuerst durch Induktion aus den gegebenen Beispiel-Transformationen ein allgemeines Verfahren finden und dieses dann explizieren. Man beachte, dass dies zwei unterschiedliche Leistungen sind. Ein Aktivitätsmuster kann auch quasi direkt als „prozedurales Wissen“ gespeichert und verwendet werden ohne expliziert zu werden [BB84] [An07]. Die Algorithmen sollten so formuliert werden, dass sie jeder der gleichen Altersgruppe versteht. Sie sollten in Phase 3 des Workshops getestet werden. Ein wichtiges Detail des Aufgabenblattes ist eine „Ideenwolke“ mit Begriffen, die eventuell für die Formulierung des Algorithmus verwendet werden können. Die Begriffe in der Wolken waren zum Teil metaphorisch (z.B. „Ballon“ für eine runde Form in Abbildung 2 links) und zum Teil sachlich beschreibend (z.B. „Fläche“). Den Teilnehmern war es freigestellt, diese Begriffe zu verwenden. Die Ideenwolken sollten nur helfen, Formulierungsschwierigkeiten zu überwinden. Die gesamte Teilnehmergruppe wurde in zwei Hälften A und B geteilt. Gruppe A erhielt andere Aufgaben als Gruppe B. Die beiden Gruppen durften in dieser Phase nicht miteinander kommunizieren. Am Ende trennten die Schüler an der gestrichelten Linie in der Mitte den oberen Teil des Blattes mit der Ideenwolke und der Grafik, die das Lösungswort zeigt, ab. Der untere Teil enthält dann nur den Algorithmus und die Ausgangssituation. .

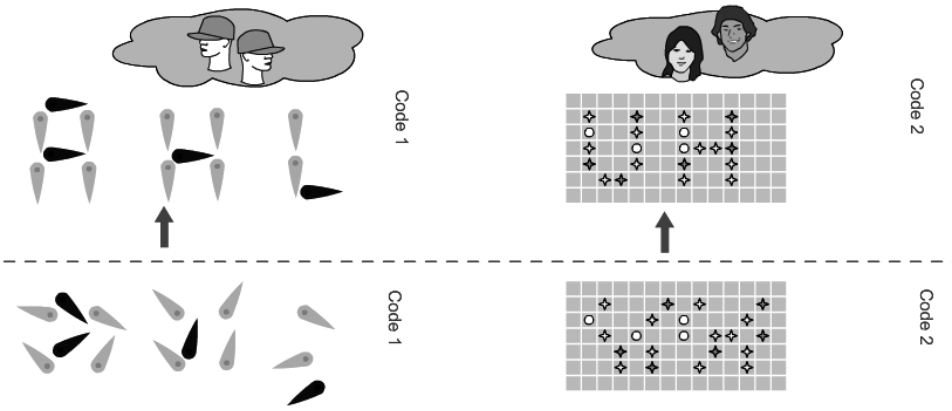

Abbildung 3: Zwei Aufgaben mit grafischer „Ideenwolke“.

In Phase 3 suchte sich jeder Schüler der Gruppe A einen Mitschüler aus Gruppe B. Die beiden tauschten Algorithmen aus und testeten, ob die Anleitung des Partners so verständlich war, dass das Lösungswort gefunden werden konnte.

Die Workshops des zweiten Typs (W2) weichen in folgenden Punkten ab: (1) Die Ideenwolken enthalten ausschließlich Metaphern - manchmal aus unterschiedlichen Domänen, so dass man sie nicht kombinieren kann. In vier der insgesamt zehn „Ideenwolken“ sind Zeichnungen anstelle von Texten (siehe Abbildung 3). Sie stellen Begriffe dar, die als Metaphern verwendet werden können.

(2) Zu Beginn von Phase 2 gibt es zunächst eine kurze Reflektion über Metaphern. Die Schüler erhalten dann explizit den Auftrag, nach Möglichkeit bei den Algorithmen eine der Metaphern aus der Ideenwolke zu verwenden

3 Ergebnisse

3.1 Dramatisierte Algorithmen verstehen

Phase 1 war bei beiden Workshop-Typen gleich. Insgesamt 119 Schülerinnen und Schüler haben die Aufgaben bearbeitet und je zwei dramatisierte und zwei nichtdramatisierte Algorithmen im Stil des obigen Beispiels (Abb. 1) nachvollzogen. Beide Varianten wurden im Schnitt zu genau 80% korrekt gelöst. Ebenso zeigte die Markierung von Aufgaben als besonders schwierig oder besonders leicht bei beiden Varianten keinen signifikanten Unterschied. Die metaphorische Ausdrucksweise war offenbar weder ein Hindernis (was aufgrund der längeren Texte hätte sein können) noch ein Vorteil.

3.2 Verwendung von Metaphern

Bei der Analyse der Algorithmen, die die Schüler selbst entwickelt hatten, wurden nur richtige oder fast richtige Algorithmen berücksichtigt.

In der ersten Workshop-Variante W1 war es freigestellt, Begriffe aus der Ideenwolke zu verwenden. Zudem waren nur einige Begriffe metaphorisch und andere beschrieben sachlich die Geometrie der Abbildung. Die Schüler hatten also völlig Freiheit in ihrer Ausdrucksweise. Im zweiten Workshop W2 wurden Schülerinnen und Schüler explizit aufgefordert, wenigstens eine Metapher aus der Ideenwolke zu verwenden. Beispiel: Zum Problem „Code 1“ (Abbildung 3 links) formulierte eine 15-jährige Schülerin in W2 folgenden Algorithmus:

„Es stehen viele Leute mit verschiedenfarbigen Kappen in jeweils drei Gruppen. Zeichne die Leute so ein, dass die Leute mit den helleren Kappen nach unten gucken und die Leute mit den dunkleren Kappen nach rechts gucken.“

Das ist ein Beispiel für einen dramatisierten Algorithmus auf der Basis einer Metapher, die in der Ideenwolke angeboten wurde. In W2 enthielten 8 von 11 überwiegend korrekten Lösung (8/11) diese Metapher. Das heißt: Fast jeder, der überhaupt in der Lage war einen passenden Algorithmus zu formulieren, hat auch die Kappen-Metapher verwendet. In W1 dagegen wurden nicht ein einziges Mal die in der Ideenwolke angebotenen Begriffe „Kappe“ oder „anschauen“ (die die gleiche Metapher unterstützen) verwendet.

In den 132 überwiegend korrekten Algorithmen aus W1 gab es insgesamt 50 Metaphern. Die 138 überwiegend korrekten Lösungen aus W2 enthielten dagegen insgesamt 286 Metaphern. Das heißt – etwas pointiert – die Schülerinnen und Schüler dieser

Altersgruppe sind durchaus in der Lage Algorithmen mit Metaphern zu formulieren, entscheiden sich aber dagegen, wenn sie die Wahl haben.

3.3 Dramatisierungen

Die 138 überwiegend korrekten Algorithmen aus W2 wurden an Hand festgelegter Kriterien pauschal dahingehend beurteilt, ob sie dramatisiert waren oder nicht. Im Prinzip sollten zwei Metaphern und eine Handlung erkennbar sein. Um die Validität des Bewertungsverfahrens zu testen, hatte eine Gruppe von acht Personen 10% der überwiegend korrekten Algorithmen auf der Basis der Kriterien und vier Prototypen beurteilt. Es ergab sich eine Übereinstimmung von 80%. Die folgenden zwei Beispiele gehören zu den Prototypen. Eine 15-jährige Schülerin schrieb als Lösung zum Problem „Code 2“ (Abbildung 3 rechts): „Die weißen Punkte sind weibliche Personen und die schwarzen Sterne sind Männer. Von links kommt ein starker Wind und bläst alle weißen Punkte ein Stück nach rechts. Auf einmal kommt Wind von rechts und bläst alle schwarzen Sterne nach links.“ Dieser Algorithmus wurde nicht als Dramatisierung gewertet. Denn die beiden Metaphern für die Sterne, die zunächst eingeführt worden sind, werden im zweiten Teil nicht verwendet um die Aktivität zu beschreiben. Dagegen ist der folgende Algorithmus einer 18-jährige Schülerin zum selben Problem dramatisiert: „Die blassen Sternchenchinesen gehen immer einen Schritt nach links während die temperamentvollen Dunkelhäutigen schon zwei Schritte nach links machen.“ Hier gibt es drei Metaphern, die in einem sinnvollen Zusammenhang stehen. Die Bewegung wird als „Schritte gehen“ beschrieben und das passt zu den Figuren, die als Metaphern für Kreisflächen und Sterne verwendet wurden. Während es in W1 fast keine Dramatisierungen gab, waren in W2 65 der 138 überwiegend korrekten Algorithmen dramatisiert. Das heißt die Teilnehmer kamen nicht von alleine auf die Idee, die Transformation der Bilder durch ein Drama zu beschreiben. Gleichwohl wirkte der Zwang zur Dramatisierung in W2 nicht als Bremse. In beiden Workshops lieferte jeder Teilnehmer im Schnitt 2,3 überwiegend korrekte Algorithmen ab.

Der Anteil der Dramatisierungen war bei den verschiedenen Problemen durchaus unterschiedlich. Zum Problem „Code 2“ (siehe Abb. 3) gab es unter den 17 überwiegend korrekten Lösungen nur drei Dramen (18%). Dagegen waren bei „Code 3“ (siehe Abb. 2) 12 der 14 überwiegend korrekten Lösungen dramatisiert (86%). Die Algorithmen der Mädchen enthielten etwa genauso viel Dramatik wie die der Jungen.

Geschlecht	Personen-zahl	Alter (SD)	Überwiegend korrekte Alg.	Dramen	Anteil dramatisiert
Mädchen	40	15,5 (1,1)	107	48	49%
Jungen	21	15,1 (1,0)	31	14	45%

Tabelle 1: Dramatisierung und Geschlecht

3.4 Eigenaktivität versus Passivität

Ein zentrales Merkmal des objektorientierten Programmierparadigmas ist die Idee des eigenaktiven Objektes, das zwar Aufträge empfangen kann, aber dann *selbst* seinen Zustand ändert. Dem gegenüber steht die Vorstellung, dass ein Agent die Ursache für Veränderung ist und passive Entitäten verarbeitet (Material und Werkzeug). Die folgenden beiden Algorithmen aus WS 2 zum Problem „Code 4“ (siehe Abbildung 2 rechts, allerdings mit anderer Ideenwolke) beinhalten eigenaktive Objekte:

„Es ist Brunftzeit. Die Schwarzpunktkröten blasen sich auf um zu imponieren. [...]“ (Schüler, 14 J.) „Auf ein paar Beeten liegen kleine Samen. Die Samen werden mit Wasser gegossen. Als die Blume wächst, entwickeln sich Wurzeln, die über das ganze Beet wuchern.“ (Schülerin, 14 J.)

Der folgende Algorithmus (zum gleichen Problem) dagegen beschreibt die Veränderung des Bildes nur als Veränderung passiver Entitäten durch einen Agenten.

„Male die Baumstämme aus, die von einem Specht heimgesucht wurden.“ (Schülerin, 17 J.)

Die Vorstellung eigenaktiver Entitäten (im Stil der OOP), wurde – je nach Problem – in sehr unterschiedlichem Ausmaß angewendet. Es konnte bei den untersuchten Beispielen keine generelle Tendenz ausgemacht werden. Zum Beispiel wurde das Vergrößern eines Elementes in manchen Fällen bevorzugt als Wachsen oder Dickwerden (eigenaktiv) und in anderen Fällen bevorzugt als Aufblasen oder Strecken (Veränderung einer passiven Entität) beschrieben.

4 Pädagogische Implikationen

Die Klassenraumaktivitäten, die in dieser Studie zum Einsatz kamen, haben auch lehrreiche Aspekte. Diese sollen nun angesprochen werden. Offenbar können Fünfzehnjährige umgangssprachliche Algorithmen zu Problemen wie in den Workshops innerhalb von wenigen Minuten formulieren und mit der Hilfe anderer Menschen auf Korrektheit und Verständlichkeit testen. Die Besonderheit gegenüber dem Programmieren ist nicht also nur das größere Entwicklungstempo (Minuten statt Stunden) sondern auch die soziale Komponente. Denn der umgangssprachliche Algorithmus ist für ein menschliches Auditorium. In den Workshops konnte beobachtet werde, dass die Teilnehmer durchaus an der Wirkung ihrer literarischen Algorithmen interessiert waren. Wird der andere mich verstehen? Wie gefällt ihm meine Formulierung?

„Schnelle“ erlebnisorientierte Übungen dieser Art könnten eine Ergänzung zu „langsamen“ konstruktiven Programmierprojekten sein. Denn bei der Entwicklung metaphorischer Algorithmen im Stil der Workshops praktizieren die Schüler „großes Programmieren im Kleinen“. Sie entwickeln plausible Systeme aus Akteuren und Aktivitäten. Genau das gleiche geschieht in der Anfangsphase einer

Softwareentwicklung, wenn eine Projektmetapher erfunden und eine Klassenstruktur designt wird.

Vorgegebene Metaphern können inspirierend sein und Kreativität provozieren. Ungewöhnliche Begriffskombinationen wie z.B. „Warze, aufblasen, Fliege, dick werden“ in der „Ideenwolke“ zu Aufgaben wurden von den Schülern als Herausforderung gesehen – nicht nur um einen Algorithmus zu erfinden sondern auch und vor allem ihn auf eigenwillige Art zu formulieren. Hier geht es nicht um eine produktbezogene *konstruktive* Kreativität wie sie z.B. Resnick mit Scratch hervorlocken will [Re07], sondern um *sprachliche* Kreativität. Auch sie ist (im Sinne Knuths) eine Facette der Programmierkunst.

Literaturverzeichnis

[An07] Anderson, J. R.: Kognitive Psychologie. 6. Auflage. Spektrum Akademischer Verlag, 2007.

[Ar69] Aristoteles: Über die Dichtkunst. Übersetzung von Friedrich Ueberweg. Berlin 1869.

[Ba05] Baumgarten, H.: Compendium Rhetoricum. 2. Auflage. Vandenhoeck und Ruprecht, Göttingen 2005.

[Be99] Beck, Kent (1999): Extreme Programming Explained. Boston u.a. (Addison Wesley).

[BB84] Berry, D.C.; Broadbent, D.E.: On the relationship between task performance and associated verbalizable knowledge. Quarterly Journal of Experimental Psychology. 36A 1984; S. 209-231.

[Cu06] Cummings, R. E.: Coding with power. Towards a Rhetoric of Computer Coding and Composition. In: Computers and Composition. Band 23 (4), 2006, S.430-443.

[De08] Dehn, M.J.: Working Memory and Academic Learning. John Wiley & Sons, Hoboken, New Jersey 2008.

[Fi02] Fishwick, P.A.: Aesthetic Programming: Crafting Personalized Software. In: Leonardo, 35/ 4, August 2002, S. 383-390.

[He02] Herbst, Claudia (2002). Blood, sweat and code: A new text, power and illiteracy in the context of gender. The Journal of Literacy and Technology, 2 (1). Online (abgerufen am 27.1.2011) http://www.literacyandtechnology.org/volume2/no1/herbst.html

[Kn84] Knuth, D. E.: Literate Programming. In: The Computer Journal 27/2, 1984, S. 97-111.

[Re07] Resnick, M.: All I really need to know (about creative thinking) I learned (by studying how children learn) in kindergarten. In: Proceedings of the 6th ACM SIGCHI conference on creativity & cognition, Wahington D.C., USA 2007, S. 1-6

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

[We25] Wertheimer, M.: Über Gestalttheorie. In: Philosophische Zeitschrift für Forschung und Aussprache, 1925, 1, S. 39-60.