Lesen im Informatikunterricht

Carsten Schulte

Didaktik der Informatik
Freie Universität Berlin
Takustr. 9
14195 Berlin
schulte@inf.fu-berlin.de

Abstract: Dass im Informatikunterricht Programme geschrieben werden, ist nicht ungewöhnlich. Ebenso werden Programme entworfen, getestet, analysiert und manchmal auch verändert und erweitert. Welche Rolle aber spielt das Lesen von Programmen?
Die These ist: Lesen ist zentraler Bestandteil der genannten Aktivitäten wie schreiben, analysieren, testen und erweitern. Lesen ist zudem Voraussetzung für das Verstehen von Programmen und – das ist der zentrale Punkt aus fachdidaktischer Sicht – Lesen ist eine recht gut beschreibbare Aktivität, sodass über den Zugriff des Lesens von Programmen lern- und leistungsdiagnostische Instrumente sowie Ansätze zu einer Theorie der Kompetenzentwicklung gefunden werden können. Solche Ansätze, ein Modell der Programmlesekompetenz sowie Anregungen für die Unterrichtspraxis werden im Artikel vorgestellt.

1 Programmieren lernen – Programmtexte lesen

In der informatikdidaktischen Literatur findet man kaum Treffer bei der Suche nach dem Begriff Lesen. Implizit jedoch wird das Lesen in verschiedenen informatikdidaktischen Ansätzen angesprochen: Im systemorientierten Ansatz wird vorgeschlagen, Systeme zu dekonstruieren. Die Autoren formulieren: „Software wird ‚interpretiert'" ([HMS99], S.161). Im informationszentrierten Ansatz gehört – ausgehend von der Feststellung, dass Information stets auf Repräsentation angewiesen ist (z.B.: [Hu99], [Br05]) – die „Interpretation von Repräsentationen" zum „inhaltlichen Kern des Schulfaches Informatik" ([Hu99], S. 168). Auch die Leitlinien informatischer Bildung der GI beziehen lesende Anteile ein: „Die Schülerinnen und Schüler eignen sich die Basiskonzepte ausgewählter Informatiksysteme durch Anwendung, Analyse, Modifikation und Bewertung an" ([GI00], S. 6). Dennoch finden sich kaum nähere Hinweise zum Lesen und Verstehen von Programmen. Nähere Angaben zum Lesen, etwa zu möglichen Lernhürden, Schwierigkeiten, Schwierigkeitsstufen, Übungen, Unterrichtsmethoden und zum Zusammenhang mit dem Schreiben von Programmen fehlen. Der vorliegende Artikel soll einen Beitrag leisten, diese Lücke zu schließen. Über den lesenden Zugang können möglicherweise didaktische Konzepte und Methoden zur Überprüfung von Lernfortschritten, Lernhürden bzw. dem aktuellen Leistungsniveau von Lernenden entwickelt werden.

Im Artikel wird nach der Diskussion verwandter Ansätze ein didaktisches Programmlesemodell vorgeschlagen, welches das verstehende Lesen von Programmtexten erklärt. Das Modell wird anschließend anhand der Analyse von Ergebnissen empirischer Studien aus dem Bereich Programmieren lernen, einigen unterrichtsmethodischen Vorschlägen für das Lesen von Quelltext und Konsequenzen für ein Kompetenzmodell auf seine Nützlichkeit untersucht.

2 Verwandte Ansätze

Verwandte Ansätze gibt es im Forschungsgebiet program comprehension, in Ergebnissen der informatikdidaktischen Forschung im Bereich Programmieren lernen sowie in der allgemeinen Theorie des Verstehens von W. Kintsch [Ki98].
Das Forschungsgebiet program comprehension (Programmverstehen, Softwarewartung) gibt es seit über 30 Jahren. Insbesondere sollen Wartungsarbeiten an (sehr großen) Softwaresystemen unterstützt werden. Forschungsansätze und -ergebnisse richten sich an Experten und nicht an Lernende (für einen Überblick siehe [MV94], [De01], [St05]). Die Arbeiten klären unter Anderem, welche Arten von Informationen aus einem Programmtext entnommen werden können bzw. müssen (etwa [Pe87], S. 298ff).
Einen Überblick zu informatikdidaktischen Forschungen zum Programmieren lernen bieten [SS89] und [RRR03]. Eine besondere Schwierigkeit betrifft das Verstehen des aus dem Programmtext resultierenden Ablaufs ([MR02] S. 55). Nach [LAJ05] fällt es Studenten schwer zu verstehen, dass jede Anweisung auf dem durch die vorangegangenen Anweisungen erzeugten Zustand operiert ([LAJ05], S.15). Dieselben Probleme stellen [RB05] auf der Schulebene fest. Für den Bereich Lesen sind insbesondere zwei Aspekte relevant: Das Verstehen der Programmausführung und das Zusammenfügen einzelner Teile zu einem sinnvollen Ganzen.
Nach Kintsch [Ki98] ist Verstehen ein regelbasierter Prozess, in dem aus einem Text Informationen extrahiert und unter Aktivierung relevanten Vorwissens und schlussfolgernden Denkens verschiedene mentale Repräsentationen gebildet werden, die letztendlich zu einem Gesamtverständnis führen ([Ki98], S.96f). Dieser Prozess führt schrittweise zu jeweils abstrakteren und von der ursprünglich wahrgenommenen Information unabhängigeren mentalen Repräsentationen ([Ki98], S.10-16). In diesem Prozess spielt das aktivierte und in das Verstehen integrierte Vorwissen eine große Rolle. Daher wird das (eigentlich holistische) Verstehen für analytische Zwecke in Textbasis (mit den extrahierten Informationen aus dem Text) und Situationsmodell (inklusive der gezogenen Schlussfolgerungen) getrennt ([Ki98], S.50).

3 Das didaktische Programmlesemodell

Das Modell soll bei möglichst geringer Komplexität helfen, Lern- und Lehrprozesse beim Erwerben von Programmierkompetenz zu analysieren, den Lernstand zu diagnostizieren sowie didaktische Konzepte zu planen, zu analysieren und zu bewerten.
Aus der Forderung nach Einfachheit folgt, so weit wie möglich auf paradigmen-, aufgaben-, domänen-, und werkzeugspezifische Aspekte des Programmverstehens zu verzich-

ten. Das Modell bezieht sich daher auf das an Schulen dominierende imperativ-objektorientierte Paradigma und den grundlegenden Prozess des verstehenden Lesens eines Programmtexts.

3.1 Verstehen

Verstehen kann als Prozess aufgefasst werden, in welchem dem vorliegenden Programmtext Bedeutung zugewiesen wird. Dazu wird beim Lesen des Programms eine interne mentale Repräsentation erzeugt, die die wesentlichen expliziten und impliziten Informationen aus dem Programmtext enthält. Als wesentlichen Aspekt des Verstehens fasse ich dabei die Fähigkeit auf, die im Programmtext enthaltene algorithmische Idee zu verstehen. Etwas abgeschwächt kann man im Falle einfacher Anfangsprogramme auch von Abläufen oder Verfahren sprechen. Verstehen bedeutet damit, sich von den wahrgenommenen Details des Programmtexts zu lösen. Erkennbar wird Verstehen beispielsweise an der Fähigkeit, das Programm mit eigenen Worten zusammenfassend zu erklären. Diese Erklärung muss qualitativ über das Nachvollziehen eines Programmablaufs hinausgehen, etwa indem die dahinterliegende algorithmische Idee erkannt wird und beschrieben werden kann. Folglich löst sich im Prozess des Verstehens die gedankliche Vorstellung von den gelesenen Einzelheiten des Programmtexts. Im optimalen Fall bleiben genau die wesentlichen Informationen erhalten, sodass auf Grundlage der so entstehenden Vorstellung das Programm erklärt werden kann. Wenn ein Programmtext verstanden wurde, dann kann man Schlussfolgerungen ziehen, Vorhersagen machen, den Programmtext in andere Darstellungen übersetzen, aber auch Details auf das Ganze

Der kommentierte Programmtext:

Die Funktion SortiereDurchEinfuegen sortiert das Array "A" mit *n* Elementen (z.B. Büchern) von klein nach groß.

```
SortiereDurchEinfuegen(A)
1  for i:=2 to n do                          {betrachte die Bücher an der Stelle i=2 bis n}
2      j:=i
3      while j ≥ 2 and A[j-1] > A[j]          {solange die Bücher A[j] und A[j-1] in der
   do                                         falschen Reihenfolge stehen...}
4          vertausche die Einträge A[j]       {...vertausche Bücher und...}
           und A[j-1]
5          j:=j-1                             {...gehe zum Bücherpaar eins weiter links}
6      endwhile
7  endfor
```

Beigefügte Erklärung, auf der Webseite oberhalb des Programmtextes präsentiert:

[1] „Wir bringen alle Bücher von links nach rechts im Regal fortschreitend an die richtige Position."
[2] „Das erste Buch bleibt zunächst an seinem Platz; dann nehmen wir das zweite Buch hinzu und vertauschen es mit dem ersten, falls es links von diesem stehen muss; dann nehmen wir das dritte Buch hinzu, vertauschen es mit dem zweiten, falls nötig, und vertauschen es anschließend gegebenenfalls mit dem ersten. Dann nehmen wir das vierte Buch hinzu, usw."
[3] „Allgemein nehmen wir also an, dass alle Bücher links im Bücherregal bereits sortiert sind; sodann nehmen wir eines hinzu und bringen es durch Vertauschen mit dem linken Nachbarn an die richtige Position."

Abbildung 1: http://www-i1.informatik.rwth-aachen.de/~algorithmus/algo2.php

beziehen, Auswirkungen von Änderungen vorhersagen und gezielt Änderungen am Programmtext vornehmen. Dies soll an einem Beispiel (Abbildung 1) erläutert werden.
Der Textauszug (Abbildung 1) ist dem Projekt Algorithmus der Woche [Al06] entnommen und beschreibt das Sortieren durch Einfügen am Beispiel eines Bücherregals. Das in Abbildung 1 gezeigte Beispiel bezieht sich auf die von den Autoren vorgelegte Erklärung des vorgestellten Algorithmus.
Verstehen umfasst damit das Verstehen der Ziele (Absatz [1]) und des Verfahrens (Absatz [2]). Das Verständnis des gelesenen Programmtexts – in Abbildung 2 durch das Oval dargestellt – kann analytisch in zwei Dimensionen getrennt werden: Die Textbasis enthält diejenigen Vorstellungen, die direkt den vorliegenden Informationen (dem Programmtext und eventuellen Eingabedaten) entnommen werden können. Das sind Vorstellungen über die konkrete Programmausführung sowie über den verallgemeinerten Ablauf. Die Textbasis enthält also das Verständnis des beschriebenen Verfahrens. Das Situationsmodell enthält darüber hinausgehende Schlussfolgerungen und das Verstehen der Ziele bzw. des Programmzwecks. Absatz [3] ist ein Beispiel für eine auf Basis des Verstehens gezogene Schlussfolgerung.
Verstehen beinhaltet insgesamt also (siehe Abbildung 2): a) relevante Aspekte der Programmausführung, b) relevante Aspekte des verallgemeinerten Ablaufs, c) über den Programmtext hinausgehende Schlussfolgerungen und d) die Beschreibung der allgemeinen algorithmischen Idee, die für unterschiedliche Eingaben gültig bzw. passend ist.

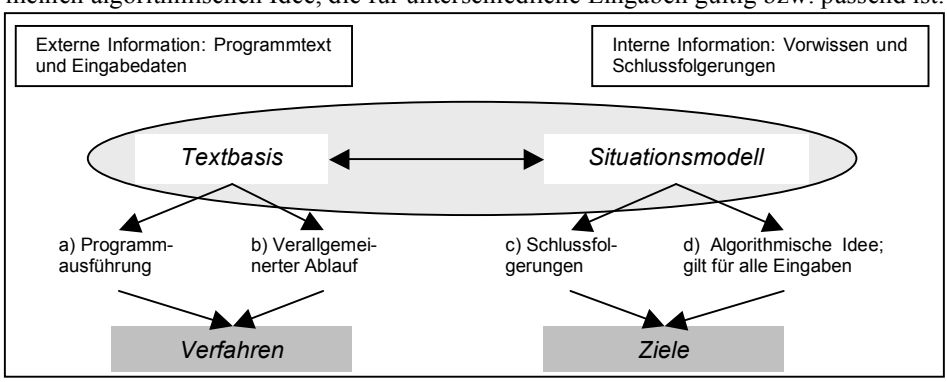

Abbildung 2: Textbasis und Situationsmodell

3.2 Der Verstehensprozess

Verstehendes Lesen von Programmtexten bezeichnet den Prozess der Wahrnehmung eines geschriebenen Quelltexts, die Extraktion relevanter Informationen, das Entwickeln einer Vorstellung von der Programmausführung sowie das Entwickeln eines generellen Verständnisses der algorithmischen Idee und der Ziele bzw. Zwecke des Texts.
Dieser Prozess kann in verschiedene Schritte unterteilt werden: Ausgehend von einzelnen Sprachkonstrukten, über kleinere Einheiten (Blöcke), deren zunehmende Integration bis zum Gesamtverstehen (siehe Abbildung 3). In Analogie zu Kintsch ([Ki98], S.101f) wird der schrittweise Verstehensprozess folgendermaßen konzipiert: Die dem Text entnommene Information wird sofort, d.h. Wort für Wort, in die mentale Repräsentation übernommen.

Auf der Konstruktebene wird die Bedeutung der einzelnen Konstrukte gebildet. Auf dieser untersten Ebene können nur Syntax und Semantik von Konstrukten sowie der Effekt der Ausführung betrachtet werden, erst ab der Blockebene sind Ziele erkennbar. Als Block kann man sich das vorstellen, was zwischen zwei Klammern oder „begin“ und „end“ steht. Unterprogramme, Methoden- und Schleifenrümpfe sind Beispiele für Blöcke. In den meisten Sprachen können Blöcke ineinander geschachtelt werden.
Blöcke übernehmen die Rolle, die Sätze in natürlichsprachlichen Texten einnehmen. Beim Lesen natürlichsprachlicher Texte wird die erste textnahe mentale Repräsentation am Satzende in das nächst abstraktere Modell integriert. Das Kurzzeitgedächtnis ist damit frei für den nächsten Lesezyklus. Es können jeweils nur wenige Einzelheiten, oder komprimierte bzw. abstrahierte Informationen übernommen werden. Daher müssen im Laufe des Verstehensprozesses die gebildeten mentalen Repräsentationen abstrakter werden.

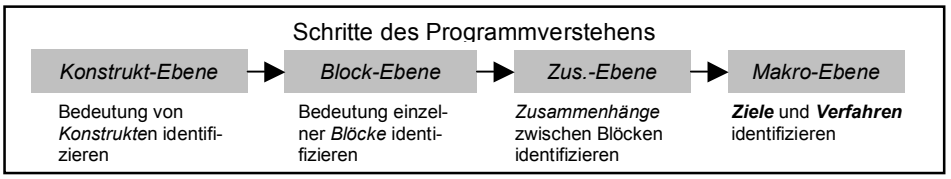

Abbildung 3: Ebenen bzw. Schritte des verstehenden Lesens eines Programms

In vielen Fällen können Blöcke benutzt werden, müssen aber nicht. Beispielsweise könnte in einem Sortierprogramm das Vertauschen zweier Variablen an verschiedenen Stellen jeweils als Folge einzelner Anweisungen geschrieben, oder in eine Methode „vertausche (a, b)“ ausgelagert werden, die jeweils aufgerufen wird. Wenn wir annehmen, dass ein Leser die Bedeutung einzelner Anweisungen, oder die einzelnen Zeilen des gelesenen Programms jeweils an einem Blockende in das nächst abstraktere Modell integrieren kann, dann ist ein Programm leichter zu verstehen, das logisch zusammengehörende Anweisungen in Blöcke verpackt. Dies liegt daran, dass die einzelnen Blöcke kürzer sind, sodass ein Leser sich nicht zu viele Einzelinformationen merken muss, bevor die Bedeutung einzelner gelesener Zeilen als Bedeutung eines Blocks zusammengefasst werden können.
Doch was bedeutet im zweiten Schritt, auf der Blockebene (siehe Abbildung 3), das Verstehen der Bedeutung eines Blocks? Die Bedeutung der Methode „vertausche (a, b)“ aus dem obigen Beispiel (Abbildung 1) kann man bereits an ihrem Namen ablesen. Die Bedeutung eines Blocks umfasst Ziel und Verfahren, während auf Konstruktebene meist noch kein Ziel erkennbar ist.
Im nächsten Schritt (Zusammenhangs-Ebene) werden Beziehungen und Zusammenhänge zwischen Blöcken in die mentale Repräsentation aufgenommen. Bleiben wir beim Beispiel oben: Dem Lesen des vertausche(a,b)-Blocks geht voran das Lesen der Bedingung im Schleifenkopf (Zeile 3 in Abbildung 1). Durch das Verknüpfen der beiden Blöcke wird verstanden, dass die Bücher nur dann vertauscht werden, wenn sie in der falschen Reihenfolge stehen (vgl. die Kommentare in Abbildung 1). Das Ziel ist also, die richtige Reihenfolge der Bücher herzustellen. Nebenbei: Die neben den Quelltextzeilen angebrachten Kommentare zeigen, dass beim Lesen diese Verstehensebenen gebildet werden; d.h. jede Quelltextzeile wird im Kontext des bis dahin Gelesenen verstanden.

Auf der obersten Ebene schließlich (Makroebene) wird der gesamte gelesene Programmtext verstanden, d.h. die Ziele sowie das Verfahren können erklärt werden.
Der Verstehensprozess läuft also bottom-up schrittweise von den unteren Ebenen zu den höheren. Eine Konsequenz dieses Verarbeitungsprozesses ist, dass Textbasis und Situationsmodell parallel konstruiert werden. Diese beiden Aspekte sind nur für Zwecke des Modells getrennt – tatsächlich entsteht eine einheitliche mentale Repräsentation.

3.3 Das Programmlesemodell

Das resultierende Programmlesemodell ist in Abbildung 4 dargestellt:

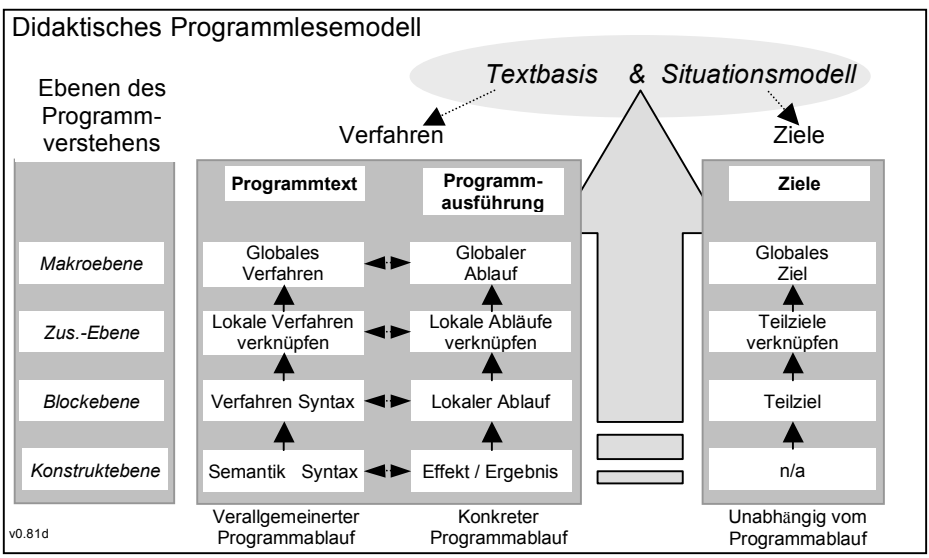

Abbildung 4: Didaktisches Programmlesemodell

Das Modell zeigt, wie auf den vier aufeinander folgenden Ebenen (linke Spalte) parallel Textbasis und Situationsmodell (oben rechts) aufgebaut werden. Die Textbasis enthält die Vorstellung des Verfahrens (mittlere Spalte). Dazu gehören die direkt dem Text entnehmbaren Informationen über den (verallgemeinerten und konkreten) Programmablauf. Wird etwa der Block j:=j+1 gelesen, wird zunächst die Syntax wahrgenommen und der Effekt (bzw. die Semantik) ermittelt: Als Semantik etwa „dekrementieren des Werts der Variablen j um 1“; der konkrete Effekt hängt vom Variablenwert ab. Mit Hilfe dieses Ablaufs wird im obigen Beispiel das Verfahren umgesetzt, die Bücherpaare von rechts nach links zu vergleichen (vgl. Kommentar in Zeile 5, Abbildung 1).
Auf den höheren Ebenen enthält die zunehmend abstraktere mentale Repräsentation nicht mehr direkte Darstellungen des gelesenen Texts, der eigentlichen Syntax, sodass dieser Aspekt nur auf Ebene 1 und 2 vorkommt. Stattdessen wird vom eigentlichen Programmtext abstrahiert und auf den gebildeten mentalen Repräsentationen operiert. Das Programmlesemodell unterscheidet in der Textbasis zwischen dem Verstehen des allgemeinen Verfahrens, wie es im Programmtext beschrieben ist, und dem Verstehen des konkreten Ablaufs, der von den konkreten Eingabedaten bestimmt wird.

Zum Situationsmodell gehören das Verstehen der algorithmischen Idee, der Ziele sowie weiterer Schlussfolgerungen (rechte Spalte in Abbildung 4). Die Integration von Textbasis und Situationsmodell stellt den schwierigen Teil dar (oben rechts). Denn hier gilt es, vom (beobachtbaren / anhand des Vorwissen rekonstruierbaren) Verhalten des Programms auf die mit dem Programmtext verbundene Absicht (Ziele) zu schließen.
Der Verstehensprozess verläuft parallel zum Lesen, also zyklisch. D.h. während des Lesens des nächsten Konstrukts werden die Stufen jeweils wieder neu durchlaufen, soweit das möglich ist. Falls etwa schon Blöcke gelesen wurden, werden beim Lesen des nächsten Blocks mögliche Zusammenhänge gebildet. Am Blockende wird jeweils das gebildete mentale Modell soweit abstrahiert, dass von der konkreten Textgestalt und einzelnen Anweisungen abstrahiert wird, sodass das sensorische Kurzzeitgedächtnis die nächsten zu lesenden Konstrukte wieder aufnehmen kann. Obwohl dieser Prozess streng bottom-up verläuft, gibt es Aspekte des intentionalen top-down-Vorgehens durch Hypothesen, die beim Lesen gebildet werden, oder durch Arbeitsaufträge, die vor dem Lesen gegeben werden. Der Verstehensprozess wird so durch aktiviertes Vorwissen bzw. Hypothesen beeinflusst. Diese Beeinflussung wird stärker, wenn Lernende die unteren Schritte (das Verarbeiten und Verstehen der Bedeutung einzelner Konstrukte) bereits automatisiert bewältigen können. Zudem werden vermutlich Teile des Programmtexts wiederholt gelesen, insbesondere von Anfängern. Damit wird das Lesen nochmals durch die bis dahin aufgebaute mentale Repräsentation beeinflusst.
Bezüglich der Kompetenzentwicklung wird angenommen, dass sich die Verstehens-Kompetenz der Lernenden von unten nach oben entwickelt, dass also die unteren Ebenen schneller automatisiert verarbeitet werden als die oberen. Ein kompetenter Leser unterscheidet sich von einem Anfänger vor allem dadurch, dass die notwendigen grundlegenden Verarbeitungsschritte stärker automatisiert ablaufen können. Zudem passieren beim Aufbau des Verstehens weniger Fehler und es kann mehr relevantes Vorwissen aktiviert werden. Dadurch bleiben mehr kognitive Ressourcen für schlussfolgerndes Denken und tiefer gehende Verarbeitungsschritte übrig.

4 Anwendungen des Modells

In diesem Kapitel werden beispielhaft Anwendungen des Modells gezeigt. Diese dienen auch als Test der Gebrauchstauglichkeit des Modells im Sinne der Erklärungskraft, inneren Stimmigkeit und externen Gültigkeit.

4.1 Analyse einer Studie zur Programmierkompetenz

Zunächst soll das Modell benutzt werden, um die Ergebnisse einer internationalen Studie zur Programmierkompetenz zu interpretieren: Kann das Modell die Ergebnisse erklären?
In der so genannten Lister-Studie [Li04] hat eine internationale Arbeitsgruppe einen Test konzipiert, der ca. 550 Studenten aus 7 Ländern vorgelegt wurde. Der Test besteht vor allem aus Lese-Aufgaben, bei denen zum Teil eine fehlende Quelltextzeile ergänzt werden musste. Insgesamt beschreiben die Autoren die Ergebnisse folgendermaßen ([Li04], S. 128): Das beste Viertel hat ein gefestigtes Verständnis der Grundlagen der Programmierung, das schlechteste Viertel habe „fundamentale Probleme“. Die 50 % in der Mitte

verstehen Schleifen und Felder (arrays). Insgesamt gelte: „Their weakness is the inability to reliably work their way through the long chain of reasoning required to hand execute code, and/or an inability to reason reliably at a more abstract level to select the missing line of code." ([Li04], S. 128). Als problematisch erweist sich, die Ergebnisse zu verstehen und Schlussfolgerungen zu ziehen ([Li04], S. 128). In einem neueren Ansatz ([Li06]) verwenden die Autoren ein theoretisches Modell, mit dem sie ihre Ergebnisse interpretieren. Der Erkenntnisgewinn dieser Herangehensweise besteht in einen Interpretationsrahmen für die Ergebnisse, aus dem heraus sich das Leistungsniveau erklären lassen kann und von dem leistungsfördernde Maßnahmen abgeleitet werden können. Die Schlussfolgerung der Autoren lautet nun, dass die Schwierigkeiten vor allem darin bestehen, einzelne Teile eines Programmtexts miteinander in Beziehung zu setzen.
Zur Erklärung der Ergebnisse anhand des Programmlesemodells: Viele Studierende haben einen Kenntnisstand, der es ermöglicht die in der Studie verwendeten Programmtexte auf der Ebene 3 (Zusammenhangs-Ebene) zu verstehen. Sie können Konstrukte verstehen, einzelne Blöcke erklären und Beziehungen verstehen. Dies gelingt jedoch nicht so gefestigt, dass sie sicher operieren. Daher schleichen sich bei anspruchvollen und versteckten Wechselwirkungen zwischen Blöcken beim Lesen Verarbeitungsfehler bzw. vorschnelle Schlussfolgerungen ein, die zu einem falschen Gesamtverstehen führen. Die didaktische Schlussfolgerung wäre, das Verstehen des Zusammenspiels zwischen Teilen eines Programms zu üben. Zusätzlich könnte man die Verarbeitungsschritte auf den niedrigeren Ebenen trainieren, damit diese stärker automatisiert ablaufen, sodass mehr kognitive Ressourcen für komplexere Verarbeitungsschritte frei bleiben.

4.2 Leseaufgaben im Informatikunterricht

In diesem Abschnitt werden Leseaufgaben für den Informatikunterricht untersucht, die in einem fachdidaktischen Seminar für Lehramtsstudierende erprobt und anhand des Programmlesemodells analysiert wurden. Bei den drei Aufgabentypen handelt es sich um: „Lesepuzzle", „Finde den Fehler" und „Lesen mit verteilten Rollen".

Lesepuzzle

Der zu lesende Quelltext wird in einzelne Zeilen oder Abschnitte aus mehreren Zeilen zerschnipselt. Die Aufgabe ist, das Quelltextpuzzle wieder richtig zusammenzusetzen. Die Lernenden bekommen jeweils ein eigenes Puzzle. Nachdem die Teile ausgepackt und auf dem Tisch verteilt sind, versuchen einige, zunächst verschiedene größere Teile zusammenzusetzen. Andere versuchen sofort, einen einheitlichen Text zu erstellen. In allen Fällen werden Papierschnipsel aufgenommen, miteinander verglichen, auf dem Tisch verschoben und ausgetauscht. Das Lösen der Aufgabe war schwerer als erwartet. Als einzige Methode wurde sie zweimal erprobt, da beim ersten Mal fast niemand in der geplanten Zeit die Lösung gefunden hat.
Die Analyse anhand des Programmlesemodells: Bei der ersten Erprobung bestand das Puzzle überwiegend aus einzelnen Zeilen, und damit sind die Ziele des Textes nicht bzw. kaum zu erkennen. Die Lernenden bekommen nur Informationen auf der untersten Ebene des Programmlesemodells (siehe Abbildung 4). Beim zweiten Versuch entsprachen die Puzzleteile eher der Block-Struktur des Programms. Dieses Mal konnte das Puzzle gelöst werden, da nun für das Zusammenfügen der Teile die Beziehungen zwischen

Blöcken geklärt werden mussten. Damit setzt diese Variante voraus, dass die Lernenden die einzelnen Blöcke verstehen.
Bewertung: Es hat sich herausgestellt, dass die meisten Studenten das Puzzle anhand einfacher Hinweise im Programmtext gelöst haben: Puzzleteile, in denen Variablen deklariert werden, gehören tendenziell eher an den Anfang, Teile mit Ausgaben für den Benutzer eher ans Ende. Obwohl die Aufgabe motivierte und die Lernenden aktivierte, hat das verwendete Puzzle nicht durchgehend die gewünschte Verarbeitung auf den Ebenen 2 und 3 gefordert. Im Ergebnis konnten trotz richtiger Lösung nicht alle die Intention des Programms erklären, haben also kein Gesamtverständnis aufgebaut. Für einen effektiven Einsatz muss die Methode also sehr sorgfältig vorbereitet werden (bis hin zum Anfertigen der Teile: Die Schnittkanten dürfen nicht verraten, wie die einzelnen Teile zusammengehören).

Finde den Fehler

Bei dieser Methode wird ein den Lernenden an sich bekanntes Programm in veränderter Fassung der gesamten Klasse präsentiert. Aufgabe ist, die Fehler möglichst schnell zu finden und zu erklären. Dies können syntaktische und semantische Fehler sein. Für jeden gefundenen und richtig erklärten Fehler gibt es Punkte. Sofort mit der Präsentation des veränderten Quelltextes versuchen die Lernenden, Fehler zu finden und sich möglichst schnell zu melden. Einige rufen sogar die Lösung laut in die Klasse.
Die Analyse anhand des Programmlesemodells: Die Methode bezieht sich bei syntaktischen Fehlern auf die unterste Ebene. Die Erklärung semantischer Fehler setzt voraus, die Verbindung des einzelnen Blocks zum Gesamtziel erklären zu können. So werden auch die Ebenen 3 und stärker noch Ebene 4 trainiert. Die Erklärungen unterstützen zudem die Verbindung von Textbasis und Situationsmodell.
Bewertung: Den Studenten hat diese Methode sehr gut gefallen; vermutlich auch aufgrund des Wettbewerbscharakters durch die Punktevergabe. Allerdings gab es daher auch störende Zwischenrufe, wenn jemand einen Fehler zwar identifiziert hat aber nicht gleich erklären kann. Diese hereingerufenen Erklärungen haben den Verstehensprozess unterbrochen. Da recht einfach Fehler bzw. interessante algorithmische Abwandlungen eingebaut werden können, ist diese Methode flexibel und recht einfach vorzubereiten.

Lesen mit verteilten Rollen

Bei dieser Methode handelt es sich um ein vereinfachtes Rollenspiel: Verschiedene Sprecher lesen einzelne Textabschnitte vor und müssen dabei die Programmausführung anhand vorgegebener Eingabedaten inszenieren. Im Seminar gab es zudem die Rolle, an der Tafel eine Wertebelegungstabelle parallel zum vorgelesenen (bzw. durchgespielten) Programmtext mitzuführen. Die verschiedenen Rollen wurden wie folgt verteilt: Pro Block liest ein Sprecher alle Vergleichoperationen (und das Resultat mit den aktuellen Werten); der andere den Rest. Bei jedem erneuten Methodenaufruf lesen neue Sprecher.
Die Analyse anhand des Programmlesemodells: Da jeweils ein Programmblock gelesen und anschließend die Ausführung erläutert wird, trainiert diese Methode den Aufbau der Textbasis, insbesondere die Verbindung zwischen Programmausführung und Programmtext. Durch das wiederholte Aufrufen derselben Methode mit verschiedenen Werten wird auch der Aufbau des Situationsmodells gefördert, da zunehmend die Ziele der einzelnen Blöcke und ihre Beziehungen untereinander deutlich werden.

Bewertung: Die Durchführung ist recht zeitaufwändig, weil der Quelltext manuell ausgeführt wird. Zudem ist jeweils nur ein Teil der Gruppe aktiv, auch wenn die Zuhörer die Aufgabe bekommen haben, die Arbeit der Sprecher zu prüfen. Daher wirkt die Methode mitunter zäh, allerdings wird durch Sprecherwechsel und zunehmend schnelleres Lesen bei wiederkehrenden Passagen deutlich, dass das Programm besser verstanden wird. Bei der Erprobung im Seminar wechselten einige Leser z.B. unbeabsichtigt die Ebene und lasen nicht mehr die Anweisungen vor, sondern erklärten sofort Verfahren und Ziele des Textabschnitts; als wenn sie z.B. in Abbildung 1 nicht mehr nur den Quelltext, sondern die Kommentare sehen und diese lesen würden.

Leseaufgaben und Programmlesemodell

Die Diskussion der verschiedenen Unterrichtsmethoden mit Hilfe der Begrifflichkeit des didaktischen Programmlesemodells sollte zeigen, wie dieses zur Analyse und Planung von Aufgaben genutzt werden kann. Es liefert Erklärungen bzw. Thesen für den Lerneffekt der Methoden sowie Hinweise für die Gestaltung und Bewertung. Beispielsweise ist nach dem Modell das Puzzeln möglicherweise doch nicht sehr lernwirksam, obwohl alle Lernenden aktiv sind. Die Effektivität der Methode hängt stark von der Vorbereitung (Zuschnitt der Puzzle-Teile) ab. Dagegen scheint das Lesen mit verteilten Rollen zwar recht langwierig und für die meisten Lernenden eher passiv zu sein, unterstützt aber das Aufbauen des Programm-Verständnisses, da die einzelnen Verstehenselemente nacheinander sichtbar bzw. hörbar gemacht werden. Insbesondere wird der Zusammenhang zwischen Textbasis (Verfahren) und Situationsmodell (Ziel) deutlich. Die dritte vorgestellte Methode (Finde den Fehler) hängt stark von der Durchführung ab: Wenn Fehler nicht nur gefunden, sondern erklärt werden müssen, dann wird an verschiedenen (vom Lehrer vorher ausgewählten Stellen) der Zusammenhang zwischen einem Detail und dem Ganzen deutlich, etwa zwischen Block und Gesamtziel, Zusammenhänge zwischen Blöcken oder Zeile/Konstrukt und Block oder Textbasis und Situationsmodell.
Des Weiteren kann das Modell auch zum Erfinden bzw. zum Anpassen der Lesemethoden für die konkrete Unterrichtssituation herangezogen werden.
Insgesamt können die Lesemethoden in Präsentations-, Inszenierungs- und Transformationsmethoden unterteilt werden. Präsentationsmethoden präsentieren den Text auf ungewöhnliche Art, etwa als Puzzle. Inszenierungsmethoden erwecken den Text zum Leben, indem sie ihn mit der Programmausführung verbinden. Beispiele sind die oben vorgestellte Methode des Lesens mit verteilten Rollen, Rollenspiele u.ä. . Transformationsmethoden regen zum genauen Lesen an, indem sie das Ändern bzw. Ergänzen der Darstellung erfordern. Beispielsweise das Übersetzen des Quelltextes in grafische Darstellungsformen, die Methode Finde den Fehler oder Kommentierungs-Methoden, bei denen Kommentare zum Quelltext ergänzt, benotet, verändert werden müssen. Das Programmlesemodell kann zur Ausgestaltung der Unterrichtsmethoden genutzt werden. So könnte es helfen, Kriterien für Kommentare zu finden (sollen Kommentare sich auf die Textbasis oder das Situationsmodell beziehen?) oder zu überlegen, ob auch ein Kommentarpuzzle (Ordne die Kommentare ihren Quelltextzeilen zu) eine sinnvolle Methode wäre.

4.3 Programmtexte lesen und Programmierkompetenz

Eine weitere interessante Anwendung ist der Bereich der Kompetenzmessung, der bereits im Abschnitt 4.1 gestreift wurde. Dort wurden die Ergebnisse eines (zugespitzt formuliert) „Programmlese-Tests“ anhand des Modells interpretiert. Es scheint nun nahe zu liegen, die Stufen des Leseprozesses (siehe Abbildung 3) als Kompetenzstufen zu benutzen. Dabei würde jedoch die Schwierigkeit des gelesenen Programmtextes zu wenig beachtet werden – denn das Verstehen eines Programmtexts hängt natürlich von seiner Komplexität ab (sowie von weiteren Faktoren wie der Vertrautheit des Lesenden mit entsprechenden Algorithmen oder Problembereichen). Das Programmlesemodell beschreibt also keine Kompetenzstufen, sondern den Verstehensprozess beim Lesen eines Programmtexts. Ein Kompetenz(stufen-)modell müsste beschreiben, bis zu welcher Komplexität Programme verstanden werden können.
Das Programmlesemodell ließe sich nutzen, um Hinweise für die Komplexität von Programmen abzuleiten. Beispielsweise bestehen simple Programme aus einem oder wenigen Blöcken, etwas anspruchsvollere ggf. aus mehreren Blöcken, welche direkt verbunden sind. Weitere Komplexität würde durch komplexere Zusammenhänge zwischen Blöcken entstehen, etwa durch geschachtelte Blöcke usw. Die genauen Komplexitätsstufen eines solchen Modells würde man empirisch festlegen. Dazu notwendige Aufgaben ließen sich vorab anhand des Modells konstruieren. Ein Beispiel für eine solche Herangehensweise findet sich in [BS06].
Ebenso ließe es sich zur Lerndiagnostik nutzen, um anhand des Lernstands geeignete Fördermaßnahmen zu entwickeln (vgl. die Argumentationsweise am Ende von Abschnitt 4.1). Ausgehend vom Lesen des Quelltexts zeigt das Lesemodell Prozesse und Dimensionen, die zum Verstehen führen. Diese ermöglichen es, gezielt diejenigen Kenntnisse und Fähigkeiten zu trainieren, die nach den Ergebnissen der Forschung zum Programmieren Lernen besondere Probleme bereiten: Das Verstehen der Programmausführung und das Zusammenfügen einzelner Teile zu einem sinnvollen Ganzen (vgl. Abschnitt 0).

5 Schlussbemerkung

Insgesamt spielt die Programmausführung eine entscheidende Rolle beim Verstehen von Programmtexten. Aussagen über Programmtexte beziehen sich in vielen Fällen tatsächlich auf dessen Ausführung und nicht auf den Text im eigentlichen Sinne. Das Schwierige am Verstehen eines Programms ist dementsprechend, den Zusammenhang zwischen den verschiedenen möglichen konkreten Programmausführungen und der gemeinsamen allgemeinen Idee herstellen zu können (vgl. Abbildung 2: Textbasis und Situationsmodell). Es scheint wenig sinnvoll zu versuchen, zunächst etwa die erste Stufe zu lehren, dann die zweite usw. Dies liegt daran, dass Ziele nicht auf der ersten Ebene (bzw. im ersten Verstehensschritt) sichtbar werden. Für das sinnvolle Lernen aber ist die Kenntnis der Programmziele unerlässlich. Ebenso wie das Schreiben eines Programmtexts auf die spätere Ausführung des Programms zielt, ist auch das Lesen eines Programmtexts auf die Programmausführung bezogen – es kann sogar als integraler Bestandteil des Schreibens gesehen werden. Nicht zuletzt deshalb verwenden empirische Studien zur Programmierkompetenz (vgl. das Beispiel aus Abschnitt 4.1) Leseaufgaben.

Literaturverzeichnis

[Al06] Fakultätentag Informatik: Webseite Algorithmus der Woche: http://www-i1.informatik.rwth-aachen.de/~algorithmus/index.php (Letzter Zugriff April 2007)

[BS06] Bennedsen, J.; Schulte, C.: A Competence Model for Object-Interaction in Introductory Programming. In: PPIG 2006. http://www.ppig.org/workshops/18th-programme.html

[Bo89] du Boulay, B. (1989). Some difficulties of learning to program. In: Soloway, E. and Spohrer, J. C. (Eds): Studying the novice programmer, S. 57-73

[Br05] Breier, Norbert: Informatik im Fächerkanon allgemein bildender Schulen - Überlegungen zu einem informationsorientierten didaktischen Ansatz. INFOS 2005 S. 67-78

[BDW02] Burkhardt, J.-M., Detienne, F., Wiedenbeck, S.: Object-Oriented Program Comprehension. In: Empirical Software Eng., Nr. 2, 2002, S. 115-156.

[De01] Détienne, Francoise: "Software Design – Cognitive Aspects", Springer, 2001.

[ET05] A. Eckerdal and M. Thune. Novice java programmers' conceptions of object and class, and variation theory. In ITiCSE '05, 2005.

[GI00] Gesellschaft für Informatik: Empfehlungen für ein Gesamtkonzept zur informatischen Bildung an allgemein bildenden Schulen. 2000.

[Ha99] Hampel, T.; Magenheim, J.; Schulte, C.: Dekonstruktion von Informatiksystemen als Unterrichtsmethode. Zugang zu objektorientierten Sichtweisen. INFOS 1999, S. 149-164.

[Hu99] Hubwieser, P.: Informatik als Pflichtfach an bayerischen Gymnasien. INFOS 1999, S. 165-174.

[Ki98] Kintsch, W.: Comprehension. A Paradigm for Cognition. Cambridge Univ. Press, 1998.

[LAJ05] Lahtinen, E., Ala-Mutka, K., and Järvinen, H. 2005. A study of the difficulties of novice programmers. In ITiCSE 2005

[Li04] Lister, Raymond et al. A multi-national study of reading and tracing skills in novice programmers. In ITiCSE-WGR '04, 2004.

[Li06] Lister, R., Simon, B., Thompson, E., Whalley, J. L., Prasad, C. Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. In ITICSE 2006. S. 118-122.

[MV94] Van Mayrhäuser, A.; Vans, A.M.: Program Understanding – A Survey. Technical Report CS-94-120. Colorado State University 1994. (Manuskript)

[MR02] I. Milne & G. Rowe. Difficulties in Learning and Teaching Programming - Views of Students and Tutors. Education and Information Technologies, 7(1):55-66, 2002.

[Pe87] Pennington, Nancy: Stimulus structures and mental representations in experts comprehension of computer programs. Cognitive Psychology 19, 295–341, 1987.

[RB05] Ragonis, N.; Ben-Ari, M: On understanding the statics and dynamics of object-oriented programs. In SIGCSE '05, 2005, S. 226-230.

[RRR03] A. Robins, J. Rountree, and N. Rountree. Learning and teaching programming: A review and discussion. Computer Science Education, 13(2):137–172, 2003.

[SS89] E. Soloway and J. Spohrer. Studying the Novice Programmer. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1989

[SS86] Spohrer, J. C.; Soloway, E.: Novices Mistaces: Are the folk wisdom correct. In: Communications of the ACM, 29,Nr. 7. 1986, S. 624-632.

[St05] Storey, M.-A.: Theories, Methods and Tools in Program Comprehension: Past, Present and Future. In: IWPC 2005