DOI: 10.20378/irb-93220

Technische Universität Berlin

FORSCHUNGSBERICHT
aus dem
INSTITUT FÜR PSYCHOLOGIE

93/3

Der Einfluß von Lehrmaterial auf das Lösen rekursiver Programmieraufgaben Eine Fehleranalyse

Ute Schmid & Jens Gräbener

1000 Berlin 10, Dovestraße 1-5 Telefon (030) 314-22370

Zusammenfassung

In einer Untersuchung zum Erwerb rekursiver Programmiertechniken (Schmid & Ulber 1993) wurde der Einfluß von Lehrmaterialien 1) auf den Lösungserfolg bei Programmieraufgaben und 2) auf den Lerntransfer geprüft. Als Lehrmaterialien wurden dabei 1) erklärende Texte zu den Aufgaben, 2) Beispielfunktionen und 3) beide Materialien verwendet. Es zeigte sich, daß alle drei Lernbedingungen zu vergleichbarem Lerntransfer führen. Bezüglich des Lösungserfolges bei den Programmieraufgaben ergibt sich jedoch ein bedeutsamer Nachteil für die Probanden, die nur Beispielfunktionen zur Verfügung haben.

Um genaueren Aufschluß über die Wirkung von Beispielfunktionen auf das Lösen von Aufgaben zu erhalten, wurde einer Fehleranalyse durchgeführt. Dabei wird deutlich, daß Programmieranfänger noch keine adäquaten Regeln zur Verfügung haben, um Beispielfunktionen auf Aufgaben zu übertragen: Es werden häufig gerade die Anteile der Beispiele übernommen, die an die Aufgabe hätten angepaßt werden müssen, gleichbleibende Teile werden dagegen nicht als solche erkannt. Dieses Vorgehen führt zu einem hohen Anteil an semantischen Fehlern, vor allem zu nicht-terminierenden Lösungen. Dagegen produzieren Probanden, die nur mit erklärenden Texten oder beiden Materialien arbeiten häufig schnell semantisch korrekte Lösungen, die allerdings syntaktische Fehler enthalten. Lediglich die vor allem von der Beispielgruppe erzeugten Fehler stehen jedoch in einem Zusammenhang mit dem Lösungserfolg bei den Aufgaben.

1 Einleitung

Vergleicht man Beispiele mit alternativen Lehrmaterialien, so zeigt sich häufig, daß Beispiele bezüglich des resultierenden Lerntransfers zu gleichen oder besseren Ergebnissen führen (Fong, Krantz & Nisbett 1986; Reed & Bolstad 1991). Dies wird darauf zurückgeführt, daß Beispiele den Aufbau von Schemata über Probleme des neugelernten Wissensbereichs günstig beeinflussen (Novick & Holyoak 1991).

In einem auf der ACT*-Theorie (Anderson 1983) basierenden Modell zum Erwerb von Programmierfertigkeiten mit Beispielen gehen Anderson und Thompson (1989) sogar davon aus, daß eine einzige erfolgreiche Beispielanwendung zum Erwerb einer generalisierten Regel führt, die die Lösung ähnlicher Aufgaben erleichtert. Die Autoren gehen in ihrem Modell davon aus, daß die Übertragung eines Beispiels auf eine neue Aufgabe problemlos gelingt. Dagegen stellen sie Probleme, die sich bei der Generalisierung oder Diskriminierung von im Lernprozeß erworbenen Regeln ergeben, in den Mittelpunkt.

In einer Studie zum Erwerb rekursiver Programmierfertigkeiten (Schmid & Ulber 1993) wurden die Befunde, daß Beispiele zu einem mit alternativen Materialien vergleichbaren Lernerfolg führen, bestätigt: Probanden die Beispielfunktionen als Lehrmaterial zur Verfügung hatten, zeigten vergleichbare Leistungen beim Lerntransfer wie Probanden, die erklärende Texte oder beide Materialien zur Verfügung hatten. Materialien, aus denen gleiche Informationen inferierbar sind, können nach Larkin & Simon (1987) als informationsäquivalent bezeichnet werden.

Im Gegensatz zu den oben genannten Untersuchungen wurde in dieser Studie zusätzlich die Performanz bei der Aufgabenlösung betrachtet. Hier unterscheiden sich die Lehrmaterialien deutlich: Personen, die mit Beispielfunktionen arbeiteten, lösten signifikant weniger Aufgaben in der vorgegebenen Zeit, als die beiden anderen Lerngruppen. Damit sind Beispielfunktionen und erklärende Texte nach Larkin und Simon (1987) nicht komputational äquivalent. Entgegen den im Modell von Anderson und Thompson (1989) beschriebenen Annahmen, scheint beim Lernen mit Beispielen gerade das Übertragen von Beispielen auf Aufgaben die zentrale Schwierigkeit bei Anfängern zu sein.

Dieser Befund kann möglicherweise dadurch erklärt werden, daß Beispiele nur dann sinnvoll für die Lösung neuer Aufgaben genutzt werden können, wenn bereits Kriterien für den Vergleich von Beispiel und Aufgabe (*mapping*), sowie Regeln zur Anpassung des Bei-

spiels an die Aufgabe (*adaptation*) erworben wurden (vgl. auch Novick & Holyoak 1991). Um genaueren Einblick in die Wirkung von Beispielen auf die Lösung von Aufgaben zu erhalten, wurden Fehleranalysen durchgeführt. Fehlen die Kriterien für den Vergleich von Beispiel und Aufgabe, sollten Probanden nicht in der Lage sein, relevante von irrelevanten Merkmalen rekursiver Funktionen zu unterscheiden. Dies müßte sich so auswirken, daß irrelevante Übereinstimmungen von Aufgabe und Beispiel in die Aufgabe übernommen werden, relevante dagegen nicht.

2 Methode

2.1 Rekursive Aufgaben und Lernhilfen

Im Experiment (Schmid & Ulber 1993) hatten die Probanden sechs rekursive Aufgaben mit einer einfachen funktionalen Programmiersprache zu lösen. In der Logik funktionaler Programmierung ist ein Programm eine Menge von Funktionen. Eine Funktion verrechnet Eingangswerte zu einem Ergebniswert (siehe z.B. Field & Harrison 1988). Rekursive Funktionen sind solche, die sich selbst aufrufen. Eine genauere Darstellung der funktionalen Sprache und der Technik rekursiver Programmierung findet sich in Schmid und Ulber (1993).

Zu jeder der sechs rekursiven Aufgaben konnten die Probanden entweder zwei Beispielfunktionen oder zwei erklärende Texte oder beide Materialien abrufen. Die Beispielfunktionen unterscheiden sich von der Aufgabe nur in Oberflächenmerkmalen wie Funktionsname, verwendete vordefinierte Funktionen (z.B. Addition versus Multiplikation), Rückgabewerte (z.B. 0 versus 1), oder Anzahl der Eingangswerte. Die Erklärungen beschreiben die Aufgabenanforderungen in allen für rekursive Funktionen relevanten Aspekten: Abbruchbedingung, Rückgabewert, rekursiver Fall. Tabelle 1 zeigt zwei der sechs Aufgabenstellungen mit dazugehörigen Lernhilfen.

Um die Aufgaben mit Hilfe der erklärenden Texte zu lösen, ist es lediglich nötig, die vorgegebene Information in die Syntax der Programmiersprache umzusetzen. Bei der Vorgabe von Beispielfunktionen muß dagegen erkannt werden, welche Teile bei Aufgabe und Beispiel übereinstimmen und welche Teile verändert werden müssen. So ist es zur

Lösung der Aufgabe *addz* notwendig zu erkennen, daß es sich bei den Beispielen um multiplikative Verknüpfungen, bei der Aufgabe aber um eine additive Verknüpfung von Werten handelt. Für alle rekursiven Funktionen gilt, daß eine Abbruchbedingung angegeben werden muß, die durch Veränderung des relevanten Parameters im rekursiven Aufruf erreicht werden kann, anderenfalls kann die Funktion bei der Ausführung nicht beendet werden (Nicht-Termination).

TABELLE 1
Auszug aus den rekursiven Aufgaben und dazugehörigen Beispielfunktionen und erklärenden Texten

Eine endrekursive Aufgabenstellung	Lösung
Schreibe eine rekursive Funktion last, die das letzte Element einer Liste l ausgibt.	FUN last (l:natlist):nat IF EMPTY(TAIL(l)) THEN HEAD(l) ELSE last(tail(l)) FIN
Beispiel 1: Die Funktion length liefert die Länge n einer Liste l. n wird im Aufruf immer mit 0 übergeben.	Beispiel 2: Die Funktion sblast bildet die Summe x aller Elemente der Liste l ohne das letzte Element. x wird im Aufruf immer mit 0 übergeben.
FUN length(l:natlist,n:nat):nat IF EMPTY(l) THEN n ELSE length(TAIL(l),PLUS(n,1)) FIN	FUN sblast(l:natlist,x:nat):nat IF EMPTY(TAIL(l)) THEN x ELSE sblast(TAIL(l),PLUS(x,HEAD(l))) FIN

Erklärung 1: (Erklärung 2 etwas ausführlicher)

Um das letzte Element einer Liste auszugeben, muß die Liste solange verkürzt werden, bis nur noch ein Element vorhanden ist. Dieses Element wird dann ausgegeben.
Als Abbruchbedingung muß also gelten, daß die Liste nur ein Element enthält. Damit diese Bedingung erreicht werden kann, muß die Liste im rekursiven Aufruf minimiert werden.

TABELLE 1 (Fortsetzung)

Eine teil-rest-rekursive Aufgabenstellung	Lösung
Schreibe eine rekursive Funktion addz, die den Summenwert einer Zahl x mit all ihren Vorgängern bis 0 berechnet. Beispiel: x=5 ergibt 0+1+2+3+4+5=15	FUN addz(x:nat):nat IF EQUAL(x,0) THEN 0 ELSE PLUS(x,addz(MINUS(x,1))) FIN
Beispiel 1:	Beispiel 2:
Die Funktion fac liefert die Fakultät einer Zahl x:	Die Funktion pot multipliziert eine Zahl x x+1-mal mit sich selbst. Dazu wird eine Zählvariable count eingeführt, die im Aufruf immer mit 0 übergeben wird.
FUN fac(x:nat):nat IF NOT(GREATER(x,1)) THEN 1 ELSE MULT(x,fac(MINUS(x,1))) FIN	FUN pot(x:NAT,count:nat):nat IF EQUAL(x,count) THEN x ELSE MULT(x,pot(x,PLUS(count,1))) FIN
fac(0)=1 fac(1)=1 fac(4)=1*2*3*4=24	pot(0,0)=0 pot(2,0)=2*2*2=8

Erklärung 1: (Erklärung 2 etwas ausführlicher)
Um die Summe aller Zahlen von x bis 0 zu erhalten, muß 0 ausgegeben werden, wenn x gleich null ist. Sonst muß x zum Ergebnis der Funktion für x-1 addiert werden, also x + addz(x-1).

2.2 Fehlerklassen

Bei einfachen rekursiven Funktionen, wie sie in Tabelle 1 angegeben sind, ist es möglich, eine vollständige und eindeutige Beschreibung möglicher Fehler anzugeben. Lediglich der Grad der Detailliertheit von Fehlerarten ist variabel. Wir haben uns für eine relativ detaillierte Fehlerklassifikation entschieden, um möglichst genaue Aufschlüsse über Unterschiede zwischen den Lernbedingungen zu erhalten.

Dabei sind die Fehlerklassen hierarchisch gegliedert: Allgemein können Programmierfehler in semantische Fehler, Fehler bei der Typisierung und syntaktische Fehler unterschieden werden. Zudem wurde eine Fehlerklasse für solche Fälle eingeführt, bei denen die Programmierversuche so stark abweichend von der Syntax der Sprache waren, daß die Intention der Probanden nicht mehr erkennbar war. Jede dieser Hauptkategorien wurde weiter unterteilt. Vor allem für die Klasse der semantischen Fehler wurde eine sehr feine

Unterteilung vorgenommen. Die vollständige Aufstellung aller betrachteten Fehler ist Tabelle 2 zu entnehmen.

TABELLE 2
Fehlerklassen

1 Semantische Fehler	
1.1 nicht terminierende Rekursion	
1.1.1 Abbruchbedingung falsch	f1.1.1
1.1.2 Minimierung im rekursiven Aufruf falsch	f1.1.2
1.1.3 rekursive Struktur falsch	f1.1.3
1.2 falsche Funktionalität	
1.2.1 falsche Abbruchbedingung	f1.2.1
1.2.2 falsche Rückgabe beim direkten Fall	f1.2.2
1.2.3 falsche Operation im rekursiven Aufruf	f1.2.3
1.2.4 rekursiver Aufruf fehlt	f1.2.4
1.2.5 Aufruf einer unbekannten Funktion	f1.2.5
1.3 Argument- und Ergebnistypen im Kopf	
1.3.1 Anzahl der Argumente	f1.3.1
1.3.2 Argument-Typen	f1.3.2
1.3.3 Ergebnistyp	f1.3.3
2 Typfehler	
2.1 Anzahl von Argumenten bei vordef. Funktionen	f2.1
2.2 Typ eines Arguments bei vordef. Funktionen	f2.2
2.3 Anzahl von Argumenten b. Aufruf d. rek. Fkt.	f2.3
2.4 Typ eines Arguments b. Aufruf d. rek. Fkt.	f2.4
3 Syntaxfehler	
3.1 Syntaktisch falsch, aber semantisch korrekt	f3.1
3.2 Strukturfehler - IF-THEN-ELSE unvollständig - Paramter/Argumente nicht durch Komma getrennt	f3.2
3.3 Klammerfehler - Klammerpaarung geht nicht auf - keine Klammerung der Argumente	f3.3
3.4 Tippfehler	f3.4
4 Syntaktisch so fehlerhaft, daß Semantik nicht erkennbar	f1.4.1

Anmerkung.
Kurzbezeichnung der Fehler am Ende der Zeilen

2.3 Vorgehen bei der Fehleranalyse

Von den am Experiment (Schmid & Ulber 1993) beteiligten 49 Probanden, wurden die Aufgabenlösungsprotokolle von 39 Probanden ausgewertet. Dabei handelt es sich um jeweils 13 zufällig ausgewählte Probanden für die Lernbedingungen 1) Beispielfunktionen, 2) Erklärungen und 3) beide Materialien.

Jeder Lösungsversuch eines Probanden wurde entsprechend der Fehlerklassen (Tabelle 2) bewertet. Als Lösungsversuch wurde dabei jeder Zeitpunkt in der Aufgabenlösung gewertet, bei dem ein Proband versuchte, den Editor zu verlassen und das Programm auszuführen. Für die Fehleranalyse wurden zunächst alle vorkommenden Fehler vermerkt. Waren in einer Funktionseinheit (z.B. Abbruchbedingung) mehrere Fehler vorhanden, so wurde der schwerwiegendste Fehler gewertet. Die Bedeutsamkeit eines Fehlers ergibt sich dabei aus den drei Hauptfehlerklassen: Am schwerwiegendsten wurden semantische Fehler eingestuft, am zweiter Stelle stehen Fehler bei der Typisierung und an dritter Stelle syntaktische Fehler. Wurde zum Beispiel bei der Funktionseinheit "Abbruchbedingung" ein Tippfehler gemacht und eine Abbruchbedingung so gewählt, daß die Funktion nicht terminiert, so wurde lediglich die zur Nicht-Termination führende Abbruchbedingung als Fehler gewertet. Fehler, die über mehrere Lösungsversuche beibehalten wurden, wurden nur beim ersten Vorkommen gewertet.

3 Ergebnisse

3.1 Fehlerhäufigkeiten

Betrachtet man das Vorkommen der Fehlerklassen für die Gesamtstichprobe, so werden deutlich vor allem Fehler bezüglich der Funktionalität der rekursiven Aufgaben produziert (Tabelle 3). Am häufigsten wird eine falsche Operation im rekursiven Aufruf verwendet (f1.2.3, z.B. Addition statt Subtraktion von Werten) oder ein falscher Rückgabewert (f1.2.2) angegeben. Auch falsche Abbruchbedingungen (f1.2.1), die dazu führen, daß etwa der niedrigste Wert in der Berechnung nicht berücksichtigt wird, sind sehr häufig.

Leider sind auch solche Fehler zahlreich, bei denen nicht erkennbar ist, was der

Proband zu programmieren beabsichtigt (f1.4.1). Die große Zahl von Klammerfehlern zeigt, daß es für mit funktionaler Programmierung unvertraute Personen schwierig ist, die syntaktische Struktur konsequent einzuhalten.

TABELLE 3
Fehlerhäufigkeiten

Fehlerart	f[1]	Pbn[2]	Mod	Md	AM/SD[3]
Operation im rekursiven Aufruf (f1.2.3)	142	36 (92.31)	2	2	3.64(2.95)
Rückgabe beim direkten Fall (f1.2.2)	141	39 (100)	3	3	3.62(2.05)
Syntaktisch so fehlerhaft, daß Semantik nicht erkennbar (f1.4.1)	137	30 (76.92)	0	3	3.51(3.58)
Klammerfehler (f3.3)	130	36 (92.31)	3	3	3.33(2.31)
Abbruchbedingung (f1.2.1)	127	38 (97.44)	1	3	3.23(2.15)
Syntaktisch falsch, aber semantisch korrekt (f3.1)	087	30 (76.92)	1	2	2.31(2.12)
rekursiver Aufruf fehlt (f1.2.4)	075	31 (79.49)	1	1	1.92(1.69)
Minimierung im rekursiven Aufruf (f1.1.2)	067	24 (61.54)	0	1	1.72(2.03)
Strukturfehler (f3.2)	055	28 (71.80)	1	1	1.41(1.23)
Anzahl der Argumente (f1.3.1)	044	23 (58.97)	0	1	1.23(1.42)
Typ eines Arguments bei vordef. Funktionen (f2.2)	042	22 (56.41)	0	1	1.08(1.48)
Tippfehler (f3.4)	034	21 (53.85)	0	1	0.87(1.13)
Abbruchbedingung (Nicht-Termination) (f1.1.1)	033	21 (53.85)	0	1	0.85(0.93)
Typ eines Arguments bei Aufruf der rekursiven Fkt. (f2.4)	033	22 (56.41)	0	1	0.85(0.96)
Anzahl von Argumenten bei Aufruf der rekursiven Fkt. (f2.3)	028	14 (35.90)	0	0	0.72(1.30)
rekursive Struktur (f1.1.3)	027	15 (38.46)	0	0	0.69(1.03)
Ergebnistyp (f1.3.3)	025	17 (43.59)	0	0	0.64(0.87)
Argument-Typen (f1.3.2)	016	12 (30.77)	0	0	0.41(0.72)
Anzahl von Argumenten bei vordef. Funktionen (f2.1)	016	08 (20.51)	0	0	0.41(0.99)
Aufruf einer unbekannten Funktion (f1.2.5)	012	07 (17.95)	0	0	0.31(0.73)

Anmerkungen.
[1] *f* gibt die absoluten Fehlerhäufigkeiten als Summenwerte über die Protokolle der 39 Probanden und alle 6 rekursiven Aufgaben an.
[2] *Pbn* ist Anzahl der Probanden, die diesen Fehler mindestens einmal begangen haben; Prozentwert in Klammern.
[3] Die Maße der zentralen Tendenz wurden über die Fehlersummen pro Proband berechnet.

3.2 Zusammenhang von Fehlerarten mit Lernmaßen

Im folgenden wird betrachtet, welche Arten von Fehlern dazu führen, daß die Aufgaben nicht in der vorgebenen Zeit bewältigt werden können und welche Fehler ein Hinweis dafür sind, daß die Rekursion letztendlich nicht so verstanden wird, daß ein guter Lerntransfer möglich wird (Tabelle 4). In der Tabelle sind nur die signifikanten Korrelationen der Lernmaße mit den Fehlersummen wiedergegeben.

TABELLE 4
Zusammenhang von Fehlerarten und Lernmaßen[1]

Fehlerart[2]		Aufgabenlösung[3]	Lerntransfer[4]
falsche Abbruchbedingung	f1.2.1	-.344	
falsche Rückgabe beim direkten Fall	f1.2.2	-.312	
Anzahl der Argumente	f1.3.1	-.458	
Minimierung im rekursiven Aufruf falsch	f1.1.2	-.366	
falsche Operation im rekursiven Aufruf	f1.2.3	-.347	-.323
Aufruf einer unbekannten Funktion	f1.2.5	-.400	-.362
Argument-Typen	f1.3.2	-.354	-.372

Anmerkungen.
[1] n = 39.
Die Tabelle zeigt alle signifikaten Korrelationen, auf die Wiedergabe der nicht-signifikanten Korrelationen wird aus Gründen der Übersichtlichkeit verzichtet. kritische Korrelationen für df = 37: $r_{5\%} = 0.316$, $r_{1\%} = 0.407$
[2] Fehlerart ist die Anzahl der Fehler pro Proband.
[3] Aufgabenlösung ist mit 0 (nicht gelöst) und 1 (gelöst) kodiert; berechnet wurde der Punkt-Biseriale Korrelationskoeffizient.
[4] Lerntransfer wird durch Punktezahl in einem Abschlußtest zur Erfassung des Lernerfolgs angegeben; berechnet wurde der Spearman-Rang-Korrelations-Koeffizient.

Das Ergebnis zeigt, daß es gerade die semantischen Fehler sind, die sowohl Aufgabenlösung als auch Lernerfolg (Transfer) beeinflussen.

Probanden, die zahlreiche Fehler bei der Festlegung von Abbruchbedingung (f1.2.1) und Rückgabewert (f1.2.2) begehen, die Operationen eine falsche Zahl von Argumenten übergeben (f1.3.1) und die einen Parameter nicht oder so minimieren, daß die Abbruchbedingung nicht erreicht wird (f1.1.2), scheitern eher bei der Lösung der Aufgaben.

Wird häufig eine falsche Operation im rekursiven Aufruf verwendet (f1.2.3), unbekannte Funktionen aufgerufen (f1.2.5) und bereits bei der Funktionsdefinition die Datentypen der Eingangswerte falsch festgelegt (f1.3.2), so wird sowohl die Aufgabe häufig nicht bewältigt, als auch ein so geringes Verständnis der rekursiven Programmierung erworben, daß kaum

ein Lerntransfer vorhanden ist.

3.3 Unterschiede der Fehlerarten zwischen den Lernbedingungen

Im folgenden werden Unterschiede der Fehlerarten zwischen den Lernbedingungen betrachtet. Die Probanden, die nur mit erklärenden Texten arbeiten, haben lediglich eine höhere Zahl von Klammerfehlern (f3.3) und produzieren häufiger Lösungen, die zwar semantisch korrekt sind, aber noch syntaktische Fehler enthalten (f3.1; siehe Tabelle 5).

Das Vorhandensein von Beispielfunktionen erleichtert offensichtlich die Einhaltung der Syntax der Programmiersprache. Die Beispielfunktionen können als Muster für die korrekte syntaktische Struktur von Funktionen verwendet werden. Diese syntaktischen Fehler stehen jedoch in keinem Zusammenhang mit dem Erfolg bei der Aufgabenlösung und dem Lerntransfer.

TABELLE 5
Unterschiede beim Vorkommen verschiedener Fehlerarten zwischen den Lernbedingungen

Fehlerart		Lernbedingung			Kruskal-Wallis H-Test
		Beispiele	Erklärungen	beides	
Klammerfehler	f3.3	36	62	32	9.561*
Syntaktisch falsch, aber semantisch korrekt	f3.1	27	42	18	6.979*
Minimierung im rekursiven Aufruf falsch	f1.1.2	38	12	17	10.11*
falsche Abbruchbedingung	f1.2.1	60	36	31	11.11*
falsche Rückgabe beim direkten Fall	f1.2.2	55	48	38	n.s.
falsche Operation im rekursiven Aufruf	f1.2.3	62	42	39	n.s.
Aufruf einer unbekannten Funktion	f1.2.5	07	02	03	n.s.
Anzahl der Argumenttypen	f1.3.1	25	10	09	10.74*

Alle Fehler, die im Zusammenhang mit Lösungserfolg und Lerntransfer stehen, werden vor allem von den Probanden produziert, die nur Beispiele zur Verfügung haben (Tabelle 5).

Zur Interpretation dieser Fehlerarten bezüglich der eingangs aufgestellten Behauptung, daß Beispiele nicht sinnvoll genutzt werden können, wenn noch keine Kriterien für den Vergleich von Beispiel und Aufgabe erworben wurden, werden im folgenden obige Fehlerarten illustriert.

Die Minimierung des Parameters so, daß die Abbruchbedingung erreicht werden kann (f1.1.2) ist in allen Beispielen identisch zur Aufgabe. Dennoch verzichten gerade Probanden der Beispielbedingung auf die Minimierung (z.B. *addz(x)* statt *addz(minus(x,1))*). Die in einem Beispiel vorgeschlagene Abbruchbedingung (f1.2.1) wird ebenfalls häufig nicht als identisch zur Aufgabe erkannt (z.B. *if equal(x,0)* bei *addz* und *pot*).

Dagegen werden bei Aufgabe *last* häufig die beiden Parameter der Beispielfunktionen übernommen (f1.3.1), obwohl in der Aufgabe nur ein Parameter gefordert ist. Ebenso werden Rückgabe werde im direkten Fall (f1.2.2) sowie die Operation im rekursiven Aufruf (f1.2.3) aus dem Beispiel übernommen (z.B. bei Aufgabe *addz 1* statt *0* als Rückgabewert und Multiplikation *MULT* statt Addition *PLUS*). Auch wird häufig der Funktionsnamen aus dem Beispiel (f1.2.5; z.B. *fac* statt *addz*) übernommen.

Strukturtragende Teile rekursiver Funktionen, wie Minimierung und Abbruchbedingung werden also von den Probanden nicht als solche erkannt und damit nicht aus den Beispielen übernommen. Stattdessen werden variable, aufgabenspezifische Anteile aus den Beispielen abgeschrieben. Damit ist offensichtlich, daß das Anpassen von Beispielen an Aufgaben (*mapping* und *adaptation*) ein Prozeß ist, den Programmieranfänger schwierig ohne zusätzliche Hilfe (wie erklärende Texte) bewältigen.

4 Diskussion

Die Wirkung von Beispielen auf die Lösung von Aufgaben eines neu zu lernenden Problembereichs wurden anhand von Fehleranalysen genauer untersucht. Ausgangspunkt der Analyse war der Befund, daß Personen, die nur mit Beispielen arbeiteten, weniger rekursive Programmieraufgaben korrekt lösen konnten, als Probanden, die erklärende Texte oder beide Materialien erhielten (Schmid & Ulber 1993). Die geringere Lösungshäufigkeit der Beispielgruppe konnte durch die Analyse der Programmierfehler darauf zurückgeführt werden, daß häufig nicht erkannt wurde, welche Teile der Beispiele mit der Aufgabe übereinstimmen und

welche nicht. Entsprechend wurden häufig für die Aufgabe irrelavante Teile aus den Beispielen übernommen, relevante Teile des Beispiels dagegen verändert.

Diese Befunde legen nahe, daß Beispiele erst sinnvoll genutzt werden können, wenn bereits die relevanten Merkmale des neuen Problembereichs erworben wurden. Obwohl Probanden aller drei Lernbedingungen vergleichbare Transferleistungen aufweisen, konnte gezeigt werden, daß Probanden der Beispielgruppe vor allem solche Fehler produzieren, die einen Hinweis auf ein geringeres Verständnis des Problembereichs Rekursion geben.

Literatur

Anderson, J.R. (1983). *The Architecture of Cognition.* Cambridge, Mass.: Harvard University Press.

Anderson, J.R. & Thompson, R. (1989). Use of analogy in a production system architecture. In: S. Vosniadou and A. Ortony. *Similarity and Analogical Reasoning*. Cambridge, Mass.: Cambridge University Press, S. 267-297.

Field, A.J. & Harrison, P.G. (1988). *Functional Progamming*. Reading, Mas.: Addison--Wesley.

Fong, G.T., Krantz, D.H. & Nisbett, R.E. (1986). The effects of statistical training on thinking about everyday problems. *Cognitive Psychology, 18*, 253-292.

Larkin, J.H. & Simon, H.A. (1987). Why a diagram is (somtetimes) worth ten thousand words. *Cognitive Science, 11*, 65-99.

Novick, L.R. & Holyoak, K.J. (1991). Mathematical problem solving by analogy. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 17* (3), 398-415.

Reed, S.K. & Bolstad, C.A. (1991). Use of examples and procedures in problem solving. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 17* (4), 753-766.

Schmid, U. & Ulber, D. . (1993). Determinanten des Erwerbs von Programmierkompetenz. *Forschungsbericht aus dem Insitut für Psychologie, 93/1.*