Automatisierte Identifizierung der Problemlösestrategien von Programmieranfängern in der Sekundarstufe I

Ulrich Kiesmüller

Didaktik der Informatik
Universität Erlangen-Nürnberg
Martensstr. 3
91058 Erlangen
ulrich.kiesmueller@informatik.uni-erlangen.de

Abstract: Im Teilbereich der Algorithmik werden oft spezielle Lern- und Programmierumgebungen eingesetzt, um insbesondere im Informatikunterricht der Sekundarstufe I den Lernenden die Grundlagen des Programmierens zu vermitteln. In einigen deutschen Bundesländern (z. B. Bayern), werden die Programmiergrundlagen bereits in der 7. Jahrgangsstufe (12 bis 13-jährige) gelehrt, wo dann altersgerecht gestaltete Lern- und Programmierumgebungen wie Robot Karol und Kara, der programmierbare Marienkäfer, verwendet werden. Trotz all dieser Bemühungen bleiben die Leistungen der Lernenden oft hinter den Erwartungen zurück. Um eine Verbesserung der Lehr-Lern-Prozesse zu erreichen, ist ein möglicher Ansatz genauere Kenntnisse bezüglich der individuellen Vorgehensweisen von Lernenden beim Erstellen ihrer Lösungen unter Berücksichtigung der Lösungsqualität zu erhalten. Ziel des hier angesprochenen Forschungsvorhabens ist die Identifizierung und Kategorisierung verschiedener Vorgehensweisen der Lernenden, um Lern- und Programmierumgebungen und den damit gestalteten Informatikunterricht zu optimieren. Dazu müssen die eingesetzten Lern- und Programmierumgebungen um spezielle Untersuchungs- und Diagnosekomponenten erweitert werden, deren Ergebnisse künftig den Lernenden in Form individualisierter Hilfestellungen zur Verfügung gestellt werden können. Im vorliegenden Text werden Vorüberlegungen dargelegt, die Forschungsmethodik und die Konzipierung und Implementierung der Forschungsinstrumente erläutert, erste Studien beschrieben sowie deren Ergebnisse vorgestellt und diskutiert.

1 Motivation

In der Sekundarstufe I wird im Informatikunterricht die fundamentale Idee der Algorithmisierung [Sc97] unter Einsatz spezieller Lern- und Programmierumgebungen gelehrt. In diesem Zusammenhang erlernen die Schülerinnen und Schüler auch die Grundlagen des Programmierens mit Hilfe didaktisch reduzierter, textbasierter oder graphischer Programmiersprachen. In einigen Bundesländern (wie z. B. Bayern), in denen die Grundlagen der Algorithmik bereits in der 7. Jahrgangsstufe im Lehrplan verankert sind, werden altersgerecht gestaltete Lern- und Programmierumgebungen wie Robot Karol [FK04, Pa94] und Kara, der programmierbare Marienkäfer [Re03] eingesetzt. Die Ler-

nenden werden durch die Gestaltung der Lern- und Programmierumgebungen motiviert und können wegen der einfachen Erlern- und Bedienbarkeit bereits nach wenigen Unterrichtsstunden auch komplexere Aufgabenstellungen lösen. Bei Testläufen erhalten sie aber meist schon nach wenigen Lösungsschritten eine oft rein technische, nicht an die Vorgehensweise der Lernenden angepasste und somit wenig hilfreiche Systemfehlermeldung (z. B.: „Der Kara kann keinen Schritt tun – vor ihm ist ein Baum!“). Sie werden demotiviert, wechseln oft ziellos von ihrer bisherigen Vorgehensweise zu anderen Strategien und sind auf Hilfe angewiesen – in diesem Falle von der Lehrkraft. Es ist nicht wünschenswert, den Lernenden hier zu zwingen, seine Lösung exakt an die Musterlösung anzupassen. Vielmehr sollen „die Schüler dort abgeholt werden, wo sie sich befinden“. Dazu wird die Lehrkraft dann nicht nur den aktuellen Stand der Lösung betrachten, sondern auch versuchen herauszufinden, wie der Lernende dazu gelangt ist. Ziel des hier beschriebenen Forschungsprojekts ist es, die individuellen Vorgehensweisen der Lernenden automatisiert zu identifizieren und die Systemrückmeldungen an die Vorgehensweise der Lernenden anzupassen. Es besteht die begründete Hoffnung, so ihre Motivation erhalten zu können und sie zum selbstständigen Problemlösen zu befähigen. Da sie ihre Problemlösestrategie bis zur endgültigen Lösung beibehalten können, ist zu erwarten, dass sie die Lösungen besser im Gedächtnis behalten.

Um den Lernprozess zu verbessern, muss mehr über die Problemlösestrategie der Lernenden bereits während des Problemlöseprozesses (automatisiert) unter Berücksichtigung der Qualität des Lösungsversuchs herausgefunden werden. Damit dies erreicht wird, mussten den im Unterricht verwendeten Lern- und Programmierumgebungen spezielle Untersuchungs- und Diagnosemodule hinzugefügt werden, deren Ergebnisse dazu dienen, die Systemrückmeldungen an die Lernenden zu individualisieren [KB07].

2 Prozessbeobachtungsmethoden und Forschungsstand

Vorangegangene systematische Untersuchungen hinsichtlich der Vorgehensweise von Programmieranfängern fanden vorwiegend an Hochschulen statt. Hundhausen beschreibt in [Hu06], wie mit Bildschirmvideos das Vorgehen von Studenten beim Programmieren mit einer vorgegebenen Programmierumgebung analysiert werden kann. Zuerst müssen semantische Untereinheiten des gestellten Problems basierend auf theoretischen Erwägungen bestimmt werden. Die bei den folgenden Beobachtungen identifizierten Schritte der Lernenden müssen im Nachhinein in einem sehr zeitaufwändigen Verfahren manuell im Bezug auf die gefundenen Kategorien codiert und gegenüber der Zeit aufgetragen werden. Sowohl die von Hundhausen festgestellten Probleme als auch die Schwierigkeiten, die Chi in [Ch97] während seiner Analyse „verbaler“ Daten beschreibt, veranlassten den Autor der hier vorliegenden Arbeit, eine spezielle Untersuchungssoftware zu konzipieren und implementieren, die die automatische Sammlung von Untersuchungsdaten sowie deren graphische Darstellung unterstützt. Eine ähnliche Vorgehensweise wählte Schulte [Sc04], allerdings waren die Testpersonen in seiner Untersuchung Schüler (11. Jahrgangsstufe, 16 bis 17 Jahre) und das Thema war objektorientierte Modellierung.

Ein weiteres Ziel des hier beschriebenen Forschungsprojekts ist die automatische Kategorisierung der Vorgehensweise der Lernenden. Dazu muss ein weiteres Softwarewerk-

zeug entwickelt werden, das Muster in den durch die Trackingsoftware (s. u.) gesammelten Daten z. B. mit Hilfe von Methoden der Mustererkennung identifizieren kann. Neben der Untersuchung der Vorgehensweise ist auch eine Analyse der während des Problemlöseprozesses erstellten Artefakte hilfreich. Um die Qualität der Lösungsversuche zu bewerten, werden diese von der Diagnosesoftware mit der Hilfe von Testfällen, die speziell für diesen Zweck erstellt wurden, bewertet (ähnlich [Bl04]). Die Auswahl und Gestaltung der Testfälle wurde so getroffen, dass wesentliche Teile der Problemlösung mit ihnen überprüft werden können. Somit ist eine differenziertere Bewertung der Lösungsversuche möglich als nur die bipolare Aussage „komplett richtig“ und „falsch“.

3 Die Lern- und Programmierumgebung

Grundlage der Untersuchungen dieser Arbeit ist die Lernumgebung Kara [Re03], in der die Lernenden mit Hilfe endlicher Automaten einen virtuellen Marienkäfer kontrollieren können, der sich in einer schachbrettartigen Welt (s. Abb. 1, rechts) mit starren Hindernissen (z. B. Baumstümpfen) und beweglichen Objekten (z. B. Kleeblättern) befindet. Sie wurde im Jahr 2003 unter anderem dazu entwickelt, mittels rein visueller Programmierung Programmieranfängern die Kontrollstrukturen Kommando, Sequenz, Verzweigung und Wiederholung zu vermitteln. Hierbei muss aber weder die Terminologie noch die Sicht der endlichen Automaten notwendigerweise im Vordergrund stehen.

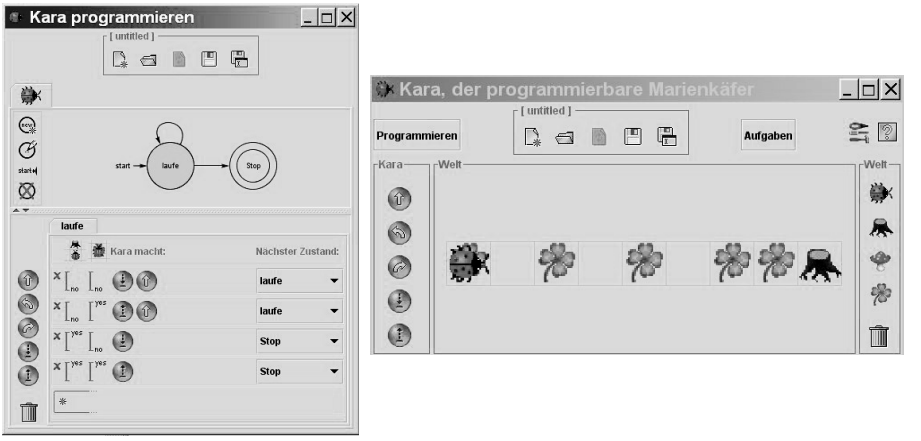

Abbildung 1: Screenshots der Kara-Lernumgebung – Programmierfenster – Modellwelt

Kara kann sich nach rechts oder links drehen, einen einzelnen Schritt nach vorne gehen (im Folgenden als *Kommandos* bezeichnet); alle Kommandos können in *Sequenzen* kombiniert werden. Außerdem besitzt Kara Sensoren, mit denen er feststellen kann, ob er z. B. auf einem Kleeblatt steht oder sich ein Baum direkt vor ihm befindet. Mit ihrer Hilfe ist es möglich, *Verzweigungen* des Programmflusses zu erreichen (s. Abb. 1, links). Durch Angabe, welcher Zustand nach Abarbeitung der Kommandosequenz eines Programmzweiges der nächste ist, werden *Wiederholungen* realisiert. Typische Aufgaben, sind z. B. durch einen „Wald“ von Bäumen zu navigieren oder Kleeblätter zu sammeln.

4 Vorüberlegungen

4.1 Lerner-System-Interaktionen

Durch theoretische Vorüberlegungen und Voruntersuchungen mit einzelnen Testpersonen, deren Vorgehen beim Lösen verschiedener typischer Kara-Aufgaben jeweils von einem Beobachter protokolliert wurde, und ergaben sich folgende für den Problemlösungsprozess relevante (also zu protokollierende) Lerner-System-Interaktionen (in den Grafiken in 6.1 später verwendeten Bezeichnungen sind kursiv hervorgehoben):

- Bearbeitung der *Zustände* – Grobstrukturierung der Problemlösung
- Bearbeitung der *Verzweigungen* und *Bedingungen* (Sensoren) – feinere Strukturierung in einzelne Teilprobleme
- Bearbeitung der *Übergänge* – bei jedem Anlegen eines Zweigs entsteht automatisch ein Übergang, so dass das erste Auftreten eines bestimmten Übergangs zur feineren Strukturierung in einzelne Teilprobleme zählt – sonst: Modellierung von Wiederholungen, Verzweigungen, Sequenzen von Sequenzen
- Bearbeitung der Sequenzen von *Kommandos* – Lösung der Teilprobleme
- Zeitpunkte der *Ausführung*sversuche, Test der (Teil-)Korrektheit der Lösungsversuche
- *Fehlermeldungen* des Systems
- Anzahlen (zeitabhängig) der oben genannten „Objekte“ und „Aktionen“

Nicht protokolliert werden u. a. Aktionen wie das „künstlerische“ Umgestalten des Aussehens von Kara und der Modellwelt sowie das Speichern von Programmversionen.

4.2 Strategien beim Problemlösen

Jedes Problem besitzt einen (unerwünschten) Anfangs- und einen (erwünschten) Endzustand – dazwischen befindet sich der Problemraum [Ma92]. Der Problemlösungsprozess wird in der Psychologie in zwei Phasen zerlegt:

- Aufspannen des Problemraumes
- Suchen eines Weges durch den Problemraum

Wird der komplette Problemraum *vor* der eigentlichen Lösung aufgespannt, ergeben sich folgende Vorgehensweisen:

- *hill climbing*
  Der Lernende versucht, den jeweils nächsten Schritt der Problemlösung möglichst optimal zu bewältigen. Durch immer wiederkehrende Erfolgskontrolle verbessert er seine Lösung wird im Laufe des Problemlöseprozesses schrittweise.
- *trial and error*
  Der Lernende versucht hier durch (manchmal zielloses) Probieren, einen Weg durch den Problemraum zu finden.

Wird hingegen der Problemraum in kleinere Unterprobleme zerlegt („divide et impera“), so ergeben sich daran anschließend zwei weitere mögliche Vorgehensweisen.

- *top down*
  In diesem Fall werden zuerst alle das gesamte Problem aufspannenden Unterprobleme aufgesucht, bevor diese gelöst werden.

- *bottom up*

Hierbei wird nun jedes einzelne identifizierte Unterproblem sofort gelöst, bevor nach weiteren gesucht wird. In strenger Form ist diese Methode nur einsetzbar, wenn die einzelnen Unterprobleme nicht verschachtelt oder verschränkt sind.

4.3 Hilfestellung für Lernende

Basierend auf den oben beschriebenen Betrachtungen wurden gestützt durch Unterrichtsbeobachtungen (verschiedener Lehrkäfte) sowie Berichten und Diskussionen bei Informatik-Fachsitzungen und Fortbildungen mögliche individualisierte Hilfestellungen für die Lernenden so gestaltet, dass diese Rückmeldungen später automatisiert von der Lernumgebung gegeben werden können unter der Voraussetzung, dass vorher die jeweilige Vorgehensweise identifiziert wurde (s. Tab.1).

<table>
<tr><th rowspan="2">Problemlöse-strategie</th><th colspan="5">Qualität des Lösungsversuchs</th></tr>
<tr><th>sehr schlecht</th><th>schlecht</th><th>mittel</th><th>gut</th><th>sehr gut</th></tr>
<tr><td>hill climbing</td><td colspan="2">Hinweis, das Problem zuerst zu strukturieren</td><td colspan="3">technische Fehlermeldung
Hilfe zur aktuellen Sequenz</td></tr>
<tr><td>trial and error</td><td colspan="3">Anleitung zum strukturierten Problemlösen</td><td colspan="2">technische Fehlermeldung
Hilfe zur fehlerhaften Sequenz</td></tr>
<tr><td rowspan="2">top down</td><td colspan="2">Verzweigungsanzahl korrekt: Hinweis auf fehlerhaften Teilzweig</td><td colspan="3" rowspan="2">technische Fehlermeldung
motivierende Bemerkung hinsichtlich guter Strukturierung der Problemlösung</td></tr>
<tr><td colspan="2">Verzweigungsanzahl falsch: Anmahnen des fehlenden Teilzweigs</td></tr>
<tr><td rowspan="2">bottom up</td><td colspan="2">Verzweigungsanzahl korrekt: Hinweis auf fehlerhaften Teilzweig</td><td colspan="3" rowspan="2">technische Fehlermeldung
Hilfe zur fehlerhaften Sequenz</td></tr>
<tr><td colspan="2">Verzweigungsanzahl falsch: Anmahnen des fehlenden Teilzweigs</td></tr>
</table>

Tabelle 1: individualisierte Rückmeldungen bei bekannter Vorgehensweise

5 Untersuchungsmethodik

5.1 Forschungsfragen

Da das Lösen algorithmischer Probleme für Neulinge auf diesem Gebiet oft mit Schwierigkeiten verbunden ist, ist es ein Ziel des hier beschriebenen Projekts, den Problemlöseprozess genauer zu erforschen, um den gesamten Lehr-Lernprozess zu verbessern. Dazu will der Autor herausfinden, welche typischen Strategien der Lernenden beim Lösen algorithmischer Probleme automatisiert unterschieden werden können, um anhand dieser

Kategorisierung die Systemrückmeldungen der Lern- und Programmierumgebungen zu verbessern und besser an die individuelle Vorgehensweise der Lernenden anzupassen.

5.2 Forschungsinstrumente

Um die in der Literatur dokumentierten Schwierigkeiten bei der Prozessbeobachtung zu vermeiden und die Möglichkeit zu haben, für die Analyse des Problemlöseprozesses relevante Daten zu erforschen und zu analysieren, wurden für diese Arbeit spezielle Forschungsinstrumente (eine Tracking- und eine Diagnosesoftware) entwickelt.

- Die Tracking-Software (hier als *TrackingKara* bezeichnet) zeichnet alle lösungsrelevanten Schritte in Abhängigkeit von der Zeit auf und ordnet sie vorher festgelegten Kategorien zu. Diese wurden – basierend auf den oben erwähnten Vorstudien – aus dem Problemlöseprozess beim zustandsorientierten Modellieren hergeleitet. Zusätzlich werden die in 4.1 aufgeführten Anzahlen in ihrer chronologischen Verteilung protokolliert und Momentaufnahmen der Lösungsversuche der Lernenden während des Lösungsprozesses gemacht, um diesen im Nachhinein rekonstruieren zu können.
- Die Diagnose-Software (hier als *EvalKara* bezeichnet) ermöglicht die Analyse der mit TrackingKara aufgezeichneten Daten in kürzerer Zeit sowie für größere Gruppen von Testpersonen. Die zeitabhängige Entwicklung aller oben erwähnten Daten ist analysierbar. Alle Daten können grafisch veranschaulicht werden, um dann von Hand weiter analysiert zu werden. Weiterhin ist eine automatische Evaluation der Qualität der Lösungsversuche unter Einbeziehung der Ergebnisse der Test-Fälle enthalten, um das Verständnis grundlegender lösungsrelevanter Ideen zu überprüfen.

5.3 Forschungsmethodik

Die softwarebasierten Forschungsinstrumente wurden basierend auf den oben beschriebenen Softwareanforderungen konzipiert und entwickelt. Um die Signifikanz der gesammelten Daten abzusichern, wurde die Software in einer Fallstudie eingesetzt, in der zehn Testpersonen mit unterschiedlichen Informatikvorkenntnissen (Anfänger bis Informatikstudent) einfache algorithmische Probleme gestellt wurden, die sie mit der Lernumgebung Kara lösen sollten. Ihre Lösungsschritte wurden mit Hilfe von TrackingKara automatisiert erfasst. Um die gesammelten Daten korrekt zu interpretieren, wurden die Testpersonen dazu angehalten, ihr Vorgehen zusätzlich mit der „thinking aloud"-Methode einem Beobachter zu erläutern. Die Auswertung führte zu den ersten Interpretationsregeln für die von der TrackingKara gesammelten Daten sowie zu weiteren jetzt umgesetzten Anforderungen für EvalKara. Abschließend wurden erste Untersuchungen mit größeren Gruppen von Probanden aus dem schulischen Bereich durchgeführt. Etwa 100 Lernende (7. Jahrgangsstufe) zweier bayerischer Gymnasien nahmen an den offiziell vom Kultusministerium genehmigten Studien teil. Da Informatik in Bayern in der 6. und 7. Jahrgangsstufe Pflichtunterricht ist, hatten zum Zeitpunkt der Untersuchungen alle Testpersonen bereits eine Einführung in grundlegende Konzepte der Informatik. Am Ende der 7. Jahrgangsstufe sieht der Lehrplan die Beschreibung von Abläufen mit Algorithmen vor. Bevor die Lernenden solche analysieren und selbst erstellen, werden sie zunächst in etwa acht Unterrichtsstunden über die Kontrollstrukturen Sequenz, (beding-

te) Verzweigung und Wiederholung unterrichtet (im vorliegenden Fall mit Einsatz der Lern- und Programmierumgebung Kara). Aufgrund der vorliegenden Randbedingungen kann angenommen werden, dass die Lernenden über einen ähnlichen Vorwissensstand verfügen. Während der Untersuchungen waren die Testpersonen dazu angehalten, die gestellten Aufgaben in Einzelarbeit (ein Lernender pro Computer) zu lösen, Kommunikation zwischen den Testpersonen war erlaubt. Innerhalb einer Unterrichtsstunde (45 Minuten) sollten die Probanden folgende drei Aufgaben aus dem Kara-Material lösen:

- A: Kara soll ein Muster aus Kleeblättern invertieren, die in gerader Linie vor ihm liegen, und stehen bleiben, wenn vor ihm ein Baum steht (s. Abb. 1).
- B: Kara soll den Eingang eines „Tunnels" aus Baumstämmen vor ihm finden und dort stehen bleiben.
- C: Kara soll den Eingang eines „Tunnels" aus Baumstämmen vor ihm finden, diesen durchlaufen und an dessen Ende stehen bleiben.

6 Erste Ergebnisse

6.1 Analysen der individuellen Vorgehensweisen

An Hand der Kategorisierung der in 4.1 beschriebenen Daten und deren zeitlichen Abfolge im Problemlöseverlauf sollen die in 4.2 aufgeführten Vorgehensweisen automatisiert identifiziert werden. Hilfe bei der Auswertung bieten so genannte Aktivitäts-Zeit-Diagramme (s. Abb. 2 bis Abb. 4) von EvalKara, die zeigen, zu welchem Zeitpunkt während des Problemlöseprozesses die Lernenden Aktivitäten aus welchem Bereich durchführen. Bestimmte Kombinationen ausgeführter und protokollierter Aktionen des Lernenden führen zu einer Einordnung der Daten in einen von vier Strategie-„Muster"-Bereichen: Werden chronologisch zuerst nur ein Teilzweig angelegt (Eintrag bei Übergang/Verzweigung), dann die zugehörigen Bedingungen gestaltet bevor dieser Zweig mit einer Sequenz von Befehlen gefüllt wird, so wird das „bottom up"-Muster zugeordnet. Wiederholt sich das Muster, so ändert sich nichts. Während der genannten Aktionsfolge auftretende *einzelne* andere Aktionen (Programmausführung) werden ignoriert. Sollten jedoch mehrere Aktionen folgen, wird entweder die Musterzuordnung geändert oder zumindest in den Ausgangszustand „kein erkennbares Muster" übergegangen. Bei direkt an die Erstellung des ersten Teilzweiges anschließende Erzeugung weiterer Teilzweige, wird eine Zuordnung zum „top down"-Muster durchgeführt, wenn anschließend die Belegung *aller* zugehörigen Bedingungen und erst abschließend das Einfügen *aller* Kommandos erfolgt. Hierbei wird zusätzlich tolerant vorgegangen, falls die Anzahl der Teilzweige um eins zu groß oder zu klein ist. Für die anderen Strategien wurden ähnliche Zuordnungsregeln für Muster festgelegt. Auf Grundlage der grafischen Darstellungen der gesammelten Daten konnten die individuellen Vorgehensweisen der Lernenden durch eine begründete Interpretation eines menschlichen Beobachters analysiert werden. Während dieser Arbeit wurde diese Zuordnung von zwei Beobachtern bei den aufgenommenen ca. 200 Sitzungen durchgeführt. Im Hinblick auf die oben erwähnten vier Strategien waren sich die Beobachter bezüglich der Zuordnung einig.

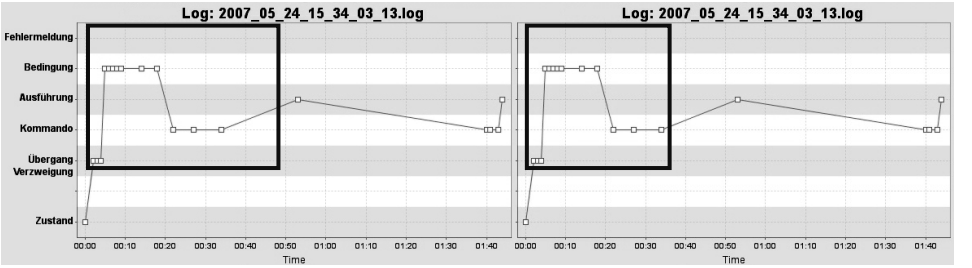

Abbildung 2: Aktivitäts-Zeit-Diagramme – Darstellung der top-down-Problemlösestrategie

Die Diagramme in Abbildung 2 stellen Beispiele dar, bei denen die Testpersonen zuerst alle notwendigen Zustände und an schließend alle Verzweigungen (und zugehörigen Zustandsübergänge) erzeugen, bevor sie am Schluss alle Zweige mit den jeweiligen Kommandos auffüllen (s. Markierung). Sie teilen den Problemraum also in kleinere Teilprobleme auf, die sie erst lösen nachdem sie den kompletten Problemraum aufgespannt haben. Dies entspricht der in Abschnitt 4.2 beschriebenen top-down-Vorgehensweise.

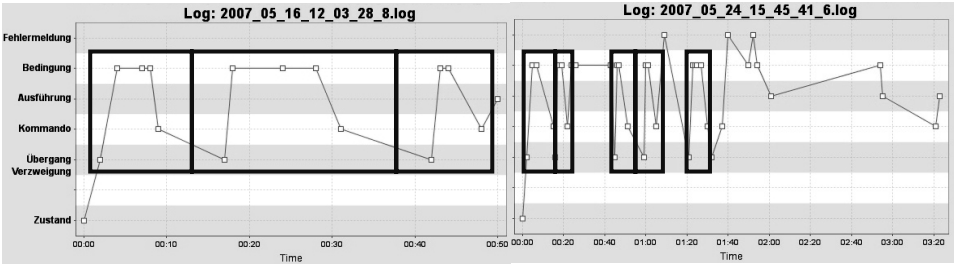

Abbildung 3: Aktivitäts-Zeit-Diagramme – Darstellung der bottom-up-Problemlösestrategie

Abbildung 3 stellt auch Lösungsverläufe dar, bei denen der gesamte Problemraum zuerst in Teilprobleme zergliedert wird. Ein Teilproblem wird hier jedoch durch Einfügen der Kommandos schon gelöst bevor der nächste editiert wird (s. Markierungen). Die Gesamtlösung entsteht, wenn der letzte Teilzweig fertig gelöst ist. Dieses Vorgehen entspricht der in Abschnitt 4.2 beschriebenen bottom-up-Methode. Musterwiederholungen zeigen die nacheinander ausgeführte Lösung der Teilzweige. Dazwischen auftretende Aktionen dienen dazu, den gerade bearbeiteten Teilzweig der Lösung zu verbessern.

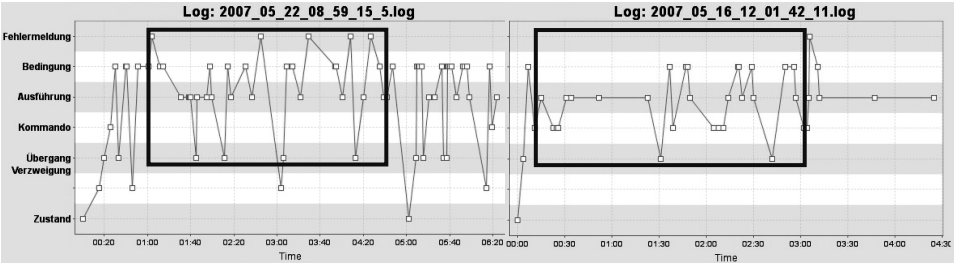

Abbildung 4: Aktivitäts-Zeit-Diagramme – trial-and-error-Strategie – hill-climbing-Methode

Einige Lernende verwenden eine reine trial-and-error-Strategie; jeder Lösungsschritt wird durch anschließende Programmausführung überprüft (s. Markierung Abb. 4 links). Die graphische Darstellung des zugehörigen Musters unterscheidet sich deutlich von denen in Abbildung 2 und 3. Außerdem ist anzunehmen, dass die auftretenden Fehlermeldungen die direkt anschließenden Schritte der Lernenden beeinflussen. Für eine diesbezügliche tiefere Analyse müssen die Fehlerarten mit in Betracht gezogen werden. Des Weiteren lässt sich die in Abschnitt 4.2 erwähnte hill-climbing-Strategie (s. Abb. 4 rechts) identifizieren – hier werden zwischen den Programmausführungen keine Systemfehlermeldungen ausgegeben, der Lernende selbst stuft die Situation von Kara als unbefriedigend ein und verbessert sein Programm schrittweise. Auch Mischungen dieser vier Problemlösemethoden sind in Form unterschiedlicher Muster in den Aktivitäts-Zeit-Diagrammen erkennbar. Zuerst muss aber die Validierung der Untersuchungswerkzeuge durch eine weitere Studie (s. Abschnitt 7) abgeschlossen werden (weshalb auch noch keine weiteren statistischen Ergebnisse betrachtet wurden), bevor eine Verfeinerung der Kategorien vorgenommen werden kann. Weiterhin lässt sich feststellen, dass Lernende, die mit einer bestimmten Methode begonnen haben ein Problem zu lösen, nach aufgetretenen Problemen bei einem Neuansatz der Lösung oft wieder dieselbe Problemlösestrategie einsetzen. Eine Ausnahme bilden hierbei Testpersonen, die nach Beginn der Lösung mit einer strukturierten Problemlösetrategie auf Grund mehrerer Fehlermeldungen und Misserfolg bei der Lösung der Aufgabe dann zur trial-and-error-Methode wechseln.

6.2 Analysen der Ergebnisse der Testfälle

Zur Beurteilung der Qualität der Lösungsversuche von Lernenden wurde die automatische Auswertung von Testfällen in EvalKara integriert. Diese wurden speziell für die einzelnen Aufgabenstellungen entwickelt und prüfen, ob zentrale Ideen der Aufgabenstellung korrekt umgesetzt wurden. Bei jedem der Testfälle erhält man entweder „success“, wenn ein Programmlauf für diesen Testfall erfolgreich war, „endless loop”, falls das Programm des Lernenden in einer Endlosschleife hängen bleibt oder die entsprechende Systemfehlermeldung, wenn der Programmlauf aufgrund eines Fehlers abbricht. Diese Rückmeldungen in Kombination mit der Information, ob das Programm das gestellte Problem löst, führen zu einer Abschätzung der Qualität des Lösungsversuches. Im Rahmen der Untersuchungen wurde die Qualität der Lösungsversuche der Lernenden außerdem von zwei Lehrkräften beurteilt (Notendurchschnitt: 2,1) und diese Ergebnisse mit den automatisiert erhaltenen verglichen. Auf diese Weise konnten noch zu verbessernde Stellen bei der Auswahl und Gestaltung der Testfälle aufgezeigt werden.

7 Zusammenfassung und Ausblick

Die Ergebnisse der hier vorgestellten Fallstudien zeigen, dass es möglich ist, verschiedene Typen von Problemlösestrategien mit Hilfe der entwickelten Forschungsinstrumente zu identifizieren, indem die gesammelten Daten sorgfältig nach Mustern durchsucht werden. Nachdem die Bewertung der Qualität der Lösungen einer Gruppe von Testpersonen durch Lehrkräfte durchgeführt wurde, wird die Korrelation dieser Ergebnisse und der automatischen Bewertungen mittels statistischer Methoden untersucht. Es soll außer-

dem überprüft werden, inwieweit die Faktoren Zeitbedarf für die Lösung, Qualität der Lösung und gewählte Vorgehensweise zusammenhängen. Der nächste Schritt ist eine gründliche Analyse der gesammelten Daten (ca. 200 Sitzungen), um in weiteren Studien (zusätzlich unterstützt durch Fragebögen) die bestehende Kategorisierung der Vorgehensweise und die Zuordnung der Muster in den Daten zu den Vorgehensweisen zu bestätigen und dann zu verfeinern.

Die grundlegenden Ideen der für Kara entwickelten Untersuchungssoftware sollen auch auf andere in Schulen verbreitete Programmierumgebungen wie z. B. Robot Karol übertragen werden, um dort ähnliche Werkzeuge zu entwickeln. Im Kontext didaktisch reduzierter imperativer Programmiersprachen muss noch geklärt werden, was die relevanten (und damit zu protokollierenden) Lerner-System-Interaktionen sind. In einem weiteren Schritt wird die Wirksamkeit der momentan vorhandenen Fehlermeldungen in den nächsten Studien mit weiteren Fragen in den erwähnten Fragebögen überprüft.

Literaturverzeichnis

[Bl04] Blumenstein, M.; Green, S.; Nguyen, A.; Muthukkumarasamy, V.: An experimental analysis of GAME: a generic automated marking environment. In Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '04). ACM Press, New York, NY, 2004; pp. 67-71. DOI=http://doi.acm.org/10.1145/1151588.1151600

[Ch97] Chi, M. T. H.: Quantifying Qualitative Analyses of Verbal Data: A Practical Guide. The Journal of the Learning Sciences, 6, 3, 1997; pp. 271-315.

[FK04] Freiberger, U.; Krško, O.: Robot Karol – Eine Programmiersprache für Schülerinnen und Schüler, 2004. URL: http://www.schule.bayern.de/karol/data/handbuch.pdf

[Hu06] Hundhausen, C. D.: A Methodology for Analyzing the Temporal Evolution of Novice Programs Based on Semantic Components. In: Proceedings of the 2006 International Workshop on Computing Education Research (ICER '06). ACM Press, New York, 2006; pp. 59-71.

[KB07] Kiesmüller, U.; Brinda, T.: Werkzeuggestützte Untersuchung der Vorgehensweisen von Lernenden beim Lösen algorithmischer Probleme. In (Eibl, C.; Magenheim, J.; Schubert, S.; Wessner, M., Hrsg.): Die 5. e-Learning Fachtagung Informatik (DeLFI 2007). Köllen, Bonn, 2007; S. 295-296.

[Ma92] Mayer, R. E.: Thinking, problem solving, cognition (2nd edition). W. H. Freeman and Company, New York, 1992.

[Pa94] Pattis, R. E.: Karel The Robot: A Gentle Introduction to the Art of Programming, 2nd Edition. John Wiley & Sons, Inc., New York, NY, 1994.

[Re03] Reichert, R.: Theory of Computation as a Vehicle for Teaching Fundamental Concepts of Computer Science. Doctoral Thesis. No. 15035. ETH Zurich, 2003. URL: http://e-collection.ethbib.ethz.ch/show?type=diss&nr=15035

[Sc04] Schulte, C.: Empirical Studies as a tool to improve teaching concepts. In (Magenheim, J.; Schubert, S., Eds.): Informatics and student assessment. Concepts of Empirical Research and Standardisation of Measurement in the Area of Didactics of Informatics. Köllen, Bonn, 2004; pp. 135-144.

[Sc97] Schwill, A.: Computer science education based on fundamental ideas. In (Passey, D.; Samways, B., Eds.): Information Technology – Supporting change through teacher education. Chapman Hall, London, 1997; pp. 285-291.