Schriftliche Hausarbeit gem. OVP v. 11. November 2003

Ein Konzept zum Einsatz von Struktogrammen als pädagogisches Werkzeug im Informatik-Unterricht

Getestet anhand einer Unterrichtsreihe zur imperativen Softwareentwicklung mit dem Java-Hamster-Modell in der 11. Jahrgangsstufe des Evangelischen Stiftsgymnasiums in Gütersloh

Schriftliche Hausarbeit
im Rahmen der Zeiten Staatsprüfung
für das Lehramt an Schulen der Sekundarstufen II und I
am Studienseminar II in Paderborn
(Gymnasium/Gesamtschule)

von

Sebastian Funke

vorgelegt zur Erstbegutachtung durch

David Tepaße

Gütersloh, den 22.05.2007

Ein Hamstergedicht

Wer irrt so spät im Kachelwald.
Es ist der Hamster und ihm ist kalt.
Ob rechts ob links er hat die Wahl,
das Korn zu suchen ist seine Qual.
So läuft er ziellos durch die Gänge,
im Nacken stets die Java-Zwaenge.
Und bis er hat das Korn gefunden,
dauert es oft viele Stunden.
Dann das Ergebnis all der Müh und Not:
Das Feld ist voll, der Hamster tot!!!

(Bild und Text entnommen aus
http://www.java-hamster-modell.de/
von Dr. Dietrich Boles)

Inhaltsverzeichnis

1 Einleitung

Ein Ziel im Informatikunterricht ist es, allgemein bildende Lerninhalte zu vermitteln. Dabei müssen einerseits bestimmte Rahmenvorgaben, wie zum Beispiel die Richtlinien für den Informatikunterricht in der Sekundarstufe II [12] und für die Sekundarstufe I [11], beim Planen und Durchführen des Unterrichts berücksichtigt werden. Andererseits gibt es aber eine gewisse Wahlfreiheit bei Unterrichtsinhalten und -methoden.
Ein Leitgedanke bei der Wahl eines Unterrichtsinhalts sollte sein, ob dieser eine fundamentale Idee in sich birgt oder nicht. Auf die Programmentwicklung bezogen gibt es sowohl allgemeinbildende Inhalte, z.B. der grundlegende Aufbau einer Kontrollstruktur, als auch Inhalte, die nicht allgemein bildend sind, wie zum Beispiel die Syntax einer konkret benutzten Programmiersprache.
Nun sind diese beiden Inhalte in der Regel eng miteinander verknüpft und bedingen sich scheinbar gegenseitig: Um die Kontrollstruktur einer while-Schleife wirklich zu verstehen, ist es hilfreich, sie auch wirklich einmal in einer konkreten Sprache zu programmieren.
Diese Problematik hat auch Peter Hubwieser bemerkt ([13]) und schlägt als Lösung dieses Dilemmas eine Implementierung von Zustandsdiagrammen vor. Das hier vorgestellte Konzept geht in eine ähnliche Richtung. Statt einer Implementierung von Zustandsdiagrammen wird jedoch eine systematische Arbeit mit Struktogrammen vorgeschlagen, welche für die programmiersprachenunabhängige Darstellung von Algorithmen um 1973 von Nassi und Shneiderman entwickelt wurden und mittlerweile unter DIN 66261 standardisiert sind ([7]).
Mit Hilfe von Struktogrammen kann sich der Schwerpunkt im Unterricht mehr auf die Vermittlung grundlegender Ideen wie z.B. der Funktionsweise einer bedingten Wiederholung konzentrieren. Gleichzeitig werden im Sinne der didaktischen Reduktion Probleme beim Beherrschen der Syntax einer konkreten Programmiersprache zunächst ausgeblendet.
Diese Vorgehensweise berücksichtigt auch die aktuelle Entwicklung in der Wirtschaft,

nämlich dem Trend zur SOA (Service Oriented Architecture), welcher ein Zusammenspiel von Programmen unterschiedlicher Sprachen über wohldefinierte Schnittstellen verstärkt ermöglichen wird, so dass die Kenntnis einer konkreten Sprache in den Hintergrund treten wird. ([8])

Eine Besonderheit des im Folgenden beschriebenen Konzeptes besteht in der Verbindung einer sehr konkreten, anschaulichen und motivierenden Ebene durch die Verwendung der Hamster-Entwicklungsumgebung und einer von der konkreten Sprache losgelösten Beschreibung eines Algorithmus durch Struktogramme.

In Konsequenz des vorliegenden Konzeptes kommt eventuell der Wunsch auf, dass die Hamster-Entwicklungsumgebung in der Lage sein sollte, ein Hamsterprogramm automatisch auch als Struktogramm darzustellen. Diesem Wunsch ist man bei anderen sehr ähnlichen Entwicklungsumgebungen bereits nachgekommen. Auch bei diesen didaktischen Programmiersprachen wird ein Einsatz von Struktogrammen vorgeschlagen ([4] und [6]).

Die vorliegende Arbeit wird erst einige Problematiken im Informatikunterricht analysieren, daraus anschließend Zielvorgaben formulieren und schließlich eine Beschreibung liefern, wie diese Ziele mit Hilfe des Einsatzes von Struktogrammen erreicht werden können. Dabei fungieren die Struktogramme nicht nur als technisches Hilfsmittel der Planung und Visualierung sondern auch als erzieherisches Hilfsmittel, um ein bestimmtes Schülerverhalten [1] zu bewirken.

Bei der Vorstellung des Konzepts wird darüberhinaus detailiert darauf eingegangen, inwiefern Struktogramme auch als ein didaktisches Hilfsmittel eingesetzt werden können, um den Schülern den Einstieg in die Softwareentwicklung mit dem Java-Hamster-Entwicklungsmodell zu erleichtern.

[1] Wenn im Verlauf dieser Arbeit von *dem Schüler* die Rede ist, so sei damit selbstverständlich ebenso *die Schülerin* gemeint.

2 Problemstellung

2.1 Qualitätsmängel in der Softwareentwicklung

In der Wirtschaft nimmt bei der Softwareentwicklung die Qualitätssicherung einen sehr hohen Stellenwert ein. Es gibt sowohl eigene Abteilungen zur Sicherstellung der Qualität der zu erzeugenden Software in der Planungsphase als auch zur Qualitätsüberprüfung bereits erstellter Software in der Testphase.
Ein gründliches Qualitätsmanagement in der Softwareentwicklung ist notwendig, da ein Softwarefehler weitreichende Konsequenzen nach sich ziehen kann. Die Folgen reichen von lästigen Anwenderproblemen, über einen Produktionsausfall bis zum Flugzeugabsturz.
Die Qualität einer Software läßt sich jedoch nicht nur an der Anzahl enthaltender Softwarefehler messen sondern auch daran, wie gut sie sich warten läßt. Schlecht kommentierter Quelltext, Code-Duplizierungen, Methoden mit Seiteneffekten, unvollständige Fallbetrachtungen und sehr spezielle Lösungen, die sich kaum wiederverwenden lassen, erschweren die Wartung einer Software. Ein Großteil der Kosten beim Einsatz einer Software entsteht nicht in ihrer Entwicklung sondern in ihrer Wartung.
Im Informatikunterricht sind die Schüler in der Regel sehr ergebnisorientiert, d.h. sie wollen möglichst schnell ein lauffähiges Programm erstellen. Wie es tut, was es tun soll, wie lesbar es für ihre Mitschüler ist, ob es wirklich alle Fälle berücksichtigt usw. ist ihnen dabei oft nicht so wichtig.
Schließlich sind die Projekte im Informatikunterricht besonders am Anfang noch klein und überschaubar, so dass sich die Konsequenzen schlecht wartbarer Programme nicht immer bemerkbar machen.
Somit hat der Lehrer das Problem, dass er einerseits seine Schüler bereits von Beginn an dazu erziehen will, dass sie Software mit hoher Qualität entwickeln, und andererseits Schwierigkeiten hat, sie dazu zu motivieren, da sich ein wichtiges Maß für die Qualität einer Software (ihre Wartbarkeit) nur schwer plausibel vermitteln läßt.

2.2 Verlust tatsächlicher Lernzeit

Im Informatikunterricht liegt eine besondere Problematik hinsichtlich der Arbeitsumgebung vor. In jedem anderen Unterricht gibt es in der Regel einen für den Unterricht vorgesehenden Arbeitsbereich. Vereinfacht kann man sagen, dass der Kunstunterricht im Kunstraum stattfindet, der Sportunterricht in der Sporthalle und der Mathematikunterricht im 'normalen' Klassenzimmer.
Im Informatik-Unterricht gibt es jedoch immer zwei grundlegend verschiedene Arbeitsbereiche. Einmal die Arbeit am Rechner und einmal die Arbeit, die nicht am Rechner stattfindet. Durch einen Wechsel zwischen einer Rechnerarbeitsphase und einer anderen Arbeitsphase kommt es daher auch zu einem Wechsel der Arbeitsbereiche und somit zwangsläufig zu Zeitverlust und Unruhe.
Auch ist eine Arbeitsphase am Rechner meist mit einer gewissen Vorlaufszeit behaftet. Die Rechner müssen eventuell erst gestartet werden, eine Anmeldung der Schüler muss erfolgen, die Schüler müssen zur Verfügung gestelltes Material herunterladen oder kopieren und so weiter.
Außerdem muss für Rechnerarbeitsphasen erfahrungsgemäß etwas mehr Zeit eingeräumt werden. Eine Rechner-Arbeitsphase von 10 min Dauer ist eher ungewöhnlich, während für eine Arbeitsphase, die nicht am Rechner geschieht, eine Zeitspanne von 10 min oft angemessen ist. Eine Rechnerarbeitsphase, die erst kurz vor Ende einer Stunde begonnen wird, ragt daher sehr wahrscheinlich in den Beginn der folgenden Stunde hinein und bringt so eine doppelte Vorlaufszeit mit sich.
Schließlich bringen Unterbrechungen der laufenden Arbeit am Rechner tedenziell mehr Zeitverlust mit sich als Unterbrechungen anderer Arbeitsphasen, da die Aufmerksamkeit der Schüler meist nur sehr schwer dem Rechner wieder entzogen werden kann.

2.3 Try-and-Error-Strategie

Sollen Schüler ein Problem am Rechner durch den Entwurf eines Algorithmus lösen, so besteht der eingeschlagende Lösungsweg oft darin, mit Try-and-Error einen passenden Algorithmus zu erstellen. Zum Beispiel wird durch Probieren herausgefunden, wie oft eine Wiederholungsanweisung durchlaufen werden muss, anstatt sich dies systematisch zu überlegen.
In einigen Situationen hat diese Strategie durchaus ihre Vorzüge. Angenommen, es liegt zum Beispiel ein Problem vor, von dem man weiss, dass es nur zwei mögliche Lösungen gibt und eine davon die richtige ist. Dann kann man unter Umständen durch Probieren schneller die richtige Lösung herausfinden als durch Nachdenken.
Liegt jedoch ein komplexeres Problem vor, so kann es sein, dass sich die gewünschte Lösung durch Probieren nicht in endlicher Zeit bestimmen läßt, da es zum Beispiel zu viele Fälle gibt. Der Versuch nun die Try-and-Error-Strategie anzuwenden, resultiert dann in einer Frustation der Schüler.
Eine weitere Folge ist ein enormer Zeitverlust. Anstatt dass die Unterrichtszeit zum Erkenntnisgewinn bzw. zum Nachdenken verwendet wird, wird getippt, kompiliert, gestartet, geschaut und dann wieder getippt.

2.4 Vorlesen von Quelltext

Die Funktionsweise eines Algorithmus kann erklärt werden, indem der Quelltext mit Beamer an die Wand projeziert und zeilenweise besprochen wird.
Bei dieser Vorgehensweise ergeben sich für die Präsentation einige Nachteile:

- Während der Präsentation des Algorithmus müssen die Mitschüler:
  - erkennen, welche Teile im Quelltext zum Algorithmus gehören.
  - sich an die gewählte Formatierung des Quelltextes gewöhnen.
  - die Semantik von der Syntax trennen.
  - sich die Bedeutung unbekannter Methoden erschließen.

- Der Quelltext eines Algorithmus geht durch die Klammersetzung, Kommentare usw. schnell über mehr als eine Bildschirmseite, so dass für die Erklärung des Algorithmus hin- und zurückgescrollt werden muss.

- Schließlich ist auch der Zeitpunkt der Präsentation des Algorithmus ein Problem. Der Algorithmus wird erst am Ende des Projekts erläutert. Das heisst, dass in einigen Fällen vielleicht versucht wird, einen nicht funktionierenden Algorithmus zu implementieren, was letztendlich zu Mißerfolgen und Frustation führt.

2.5 Einzelgängertum in der Softwareentwicklung

Besonders Schülern, die schon etwas Programmiererfahrung haben, ist es manchmal nur schwer zu vermitteln, dass das Erstellen von Softwarelösungen etwas mit Arbeiten im Team zu tun hat.
Die Tendenz besteht oft darin, dass diese Schüler eher einzelgängerisch schnell eine Lösung programmieren, anstatt zunächst gemeinsam mit anderen über mögliche Lösungen zu diskutieren bzw. gemeinsam eine Lösung zu entwerfen.
Dies ist besonders beim Einstieg in eine Programmiersprache zu beobachten, wenn die zu lösenden Probleme noch so überschaubar sind, dass man sie tatsächlich auch alleine lösen kann.

3 Zielsetzungen

3.1 Qualität durch Planung

Wird eine Software vor ihrer konkreten Implementierung gut geplant, so können etliche Softwarefehler präventiv vermieden werden, und sie läßt sich auch besser warten. Dadurch wiederum wird eine hohe Qualität der Software sichergestellt.
Tatsächlich nimmt besonders bei größeren Softwarekonzernen die Planung von Software durch sogenannte Software-Architekten einen sehr hohen Stellenwert ein.
Ein Ziel des Informatikunterrichts sollte es daher bei der Programmentwicklung von Anfang an sein, die Schüler zu einer gründlichen Planung eines Programms zu erziehen, bevor es implementiert wird.
Bei einem gründlich geplanten Programm brauchen die Schüler in der Rechnerarbeitsphase nicht mehr ihr Programm oder ihren Algorithmus grundsätzlich zu überdenken, sondern können Fehlerursachen gezielter in der Umsetzung des Algorithmus in die konkrete Programmiersprache suchen.
Die Motivation der Schüler für eine vorangehende Planung besteht dann interessanterweise darin, dass sie letztendlich oft schneller zu einer funktionierenden Lösung kommen, die ganz nebenbei eine hohe Qualität aufweist.

3.2 Effiziente Organisation der Rechnerarbeit

Ein Ziel des Informatikunterrichts muss es sein, aus der beschriebenen Situation der Rechnerarbeit die notwendigen Konsequenzen zu ziehen, um die Rechnerarbeit effizient zu organisieren und dadurch einen Zeitverlust tatsächlicher Lernzeit zu minimieren.
Dazu sollte erstens ein häufiger Wechsel zwischen Rechnerarbeit und den anderen Arbeitsphasen vermieden werden, um Zeitverlust und Unruhe zu vermeiden.
Zweitens sollte die Rechnerarbeit strikt von anderen Arbeitsphasen getrennt werden, um unnötige Unterbrechungen der Rechnerarbeit zu vermeiden.

Drittens sollte erst eine Nicht-Rechnerarbeitsphase und dann eine Arbeitsphase am Rechner durchgeführt werden, um sicherzustellen, dass den Schülern die Aufgabenstellungen für die Rechnerarbeit klar sind und dadurch Unterbrechungen der Rechnerarbeit vermieden werden.
Sollte schließlich die Wahl bestehen, eine Rechnerarbeitsphase innerhalb einer Schulstunde oder über zwei Schulstunden verteilt stattfinden zu lassen, so ist die erste Möglichkeit vorzuziehen, um die doppelte Vorlaufszeit zu vermeiden.

3.3 Einsatz und Training algorithmischen Denkens

Das Ziel im Unterricht sollte es sein, den Entwurf eines Algorithmus der Rechnerarbeitsphase vorangehen zu lassen, um das algorithmische Denken zu trainieren und eine Try-and-Error-Vorgehensweise beim Entwurf eines Algorithmus am Rechner zu vermeiden.
Dies kann zum Beispiel geschehen, indem der Algorithmus zunächst systematisch auf einem Blatt Papier entwickelt und anschließend am Rechner implementiert wird. Durch die gründliche Vorüberlegung des Algorithmus erfüllt die anschließende Implementierung am Rechner den Zweck einer Belohnung [2] und Selbstkontrolle, da der Schüler sieht, ob der von ihm entworfende Algorithmus tatsächlich funktioniert.

3.4 Sinnvolles Präsentieren von Algorithmen

Soll ein Algorithmus präsentiert werden, so geht es oft darum, anderen die Kernidee des Algorithmus zu vermitteln und nicht seine konkrete Implementierung. Dazu müssen die Schüler Möglichkeiten kennenlernen, wie sie dies tun können.
Eine wesentliche Regel dabei ist, dass man sich beim Präsentieren auf das Wesentliche

[2] Gemäß der klassischen Konditionierung wird der Schüler bei Erfolg in Zukunft von sich aus bereitwillig eine Vorplanung eines Programms vornehmen und gemäß einer Generalisierung auch auf andere Bereiche wie zum Beispiel dem Entwurf von Klassen in der OOP übertragen (siehe [9] bzw. [10])

beschränkt. Je weniger Information präsentiert wird, desto besser kann sie verarbeitet werden. Dazu kommt der Gewöhungseffekt: Je einheitlicher die Präsentationen sind, desto schneller kann man sich in einer fremden Lösung zurechtfinden.
Außerdem sollte die Präsentation eines Algorithmus nach Möglichkeit bereits *vor* seiner Implementierung stattfinden. Hat nämlich eine Gruppe beispielsweise eine nicht funktionierende Lösung erstellt, so wird sie dies während der Präsentationsphase von der restlichen Klasse bzw. dem Lehrer als Rückmeldung bekommen, und kann dann für die Implementierungsphase entweder ihre Planung korrigieren oder eine funktionierende Lösung einer anderen Gruppe übernehmen.

3.5 Softwareentwicklung als Teamwork

Das diesbezügliche Ziel im Unterricht ist es, dass die Schüler lernen, dass Softwareentwicklung in einem größeren Stil nicht nur darin besteht, mal schnell etwas zu programmieren. Vielmehr werden im Team zunächst Anforderungen an die Software geklärt und ein Lösungsentwurf erstellt. Anschließend kann die Implementierungsarbeit im Team aufgeteilt werden. Damit die Teillösungen zusammengenommen ein funktionierendes Programm ergeben, müssen im Team genaue Schnittstellen definiert und Verantwortlichkeiten verteilt werden.

4 Konzept für den Einsatz von Struktogrammen im IU

Nachdem die Problematiken und die daraus resultierenden Zielsetzungen für den IU thematisiert wurden, soll eine Verfahrensweise vorgestellt werden, wie sich diese Ziele realisieren lassen, indem man Struktogramme als Hilfsmittel verwendet.
Aus den Erörterungen zur Rechnerarbeit folgt, dass der Unterricht generell im Zweischritt verlaufen sollte. Erst werden Algorithmen in Form von Struktogrammen innerhalb einer 'theoretischen' Arbeitsphase entwickelt. Anschließend werden die erstellten Struktogramme als eine Art Bauplan für die Arbeit am Rechner verwendet. Diese Zweiteilung muss sich dabei nicht auf jede einzelne Unterrichtsstunde beziehen. Es macht im Gegenteil oft Sinn eine 'Theoriestunde' zu halten und dann eine Doppelstunde Rechnerarbeit. Dadurch, dass die Planungsphase mit einem Produkt endet, dass die Ergebnisse der Planung detailliert beinhaltet, kann der zeitliche Abstand zwischen Planung und Implementierung eines Programms auch etwas größer sein, ohne dass die Schüler zu lange brauchen, um sich wieder in das Problem hineinzufinden.
Ein Training des Algorithmischen Denkens kann durch den Einsatz von Struktogrammen erstens mit steigendem Schwierigkeitsgrad und zweitens mit unterschiedlichen Lernstrategien erfolgen. So können zum Beispiel gemäß dem Lernen am Modell nach Bandura ([10]) fertige oder halbfertige Lösungen vorgegeben werden, die von den Schülern nachzuvollziehen sind. Dadurch wird das erste Einüben neuer Kontrollstrukturen wesentlich vereinfacht und außerdem können die Schüler erfahren, was einen guten oder schlechten Algorithmus ausmacht, indem man ihnen solche exemplarisch vorgibt.
Über mehrere Abstufungen gelangen die Schüler dann dahin, dass sie selbstständig Lösungen entwerfen. Wie diese Abstufungen aussehen können, wird im Kapitel 4.2 klar, wo einige didaktische Einsatzmöglichkeiten von Struktogrammen vorgestellt

werden.

Darin wird auch nochmal darauf eingegangen, wie die Schüler mit Hilfe von Struktogrammen dazu angeleitet werden können, der Qualität einer Lösung mehr Beachtung zu schenken.

Dass sich Struktogramme gut dazu eignen, anderen die Kernidee eines Algorithmus zu vermitteln, folgt unmittelbar daraus, dass die Struktogramme in ihrer Art so ausgelegt sind, dass sie nur wichtige Information beinhalten (was der Algorithmus tut), während unwichtige Information (genaue Syntax des zugehörigen Programms) fehlt. Dadurch dass die Struktogramme außerdem erstens sehr anschaulich sind und zweitens als etwas übersichtlichere Darstellungsform einer Geschichte verwendet werden können (siehe Kapitel 4.1), eignen sie sich besonders gut als Unterstützung bei der Präsentation eines Algorithmus. Außerdem bieten sie eine Möglichkeit eine Lösung vor ihrer Implementierung zu präsentieren.

Ein Vorteil der Struktogramme gegenüber Lösungen in Form von Quelltext ist der, dass weniger Variationen im äußeren Erscheinungsbild möglich sind. Durch die simple Struktur von Struktogrammen genügen schon wenige leicht zu merkende Regeln, um ein einheitliches Aussehen der Struktogramme zu gewährleisten, wodurch die Lesbarkeit fremder Lösungen wesentlich erleichtert wird.

Um dagegen ein einheitliches Aussehen von Lösungen in Quelltextform zu ermöglichen, müsste man einen umfangreichen Regelkatalog erstellen, der darüber hinaus für handgeschriebene Lösungen eher unsinnig wäre. (Wie werden die Klammern gesetzt? Wie muss eingerückt werden? Müssen Einzelanweisungen untereinander stehen, oder dürfen sie auch hintereinander stehen? ...) Dies hätte sehr wahrscheinlich zur Folge, dass kaum ein Schüler sich an alle Regeln hält, so dass der inhaltlich gleiche Quelltext in zahlreichen visuell unterschiedlichen Varianten vorkäme, wodurch das Nachvollziehen einer fremden Lösung erschwert würde.

Inwiefern der Einsatz von Struktogrammen die Teamarbeit bei der Softwareentwicklung mit dem Hamster unterstützen kann, geht aus dem Kapitel 4.2.5 hervor.

Das vorliegende Kapitel gliedert sich nun in zwei verschiedene Unterkapitel. Zunächst wird geklärt, was überhaupt zu einem Struktogramm dazu zu zählen ist und in welcher Form die Struktogramme im Verbund mit der Hamstersprache eingesetzt werden sollten. Anschließend werden Beispiele erörtert, wie Struktogramme als didaktisches Hilfsmittel eingesetzt werden können.

4.1 Allgemeines

4.1.1 Graphische Elemente

Im Zuge dieser Arbeit werden nur die Elemente eines Struktogramms betrachtet, die für die Unterrichtsinhalte relevant waren. Unterrichtsinhalte waren:

1. elementare Anweisungen
2. Wiederholungsanweisungen mit einer Eingangsbedingung (bedingte while-Schleife)
3. Bedingte Verweigungen mit zwei Fällen (if-Bedingung mit else-Fall)
4. Prozeduren
5. boolesche Funktionen
6. Rekursionen

Bei den bedingten Verzweigungen ist anzumerken, dass immer die Angabe des else-Falls bewußt eingefordert wurde, auch wenn in diesem Fall keine Anweisung auszuführen ist, da ansonsten die Eindeutigkeit eines Struktogramms nicht garantiert werden kann.
Zur Erstellung der Struktogramme auf Arbeitsblättern kann der frei erhältliche Editor NSD verwendet werden. Eine Besonderheit der mit NSD erstellten Struktogramme ist die Art der Beschriftung. Da sämtliche Struktogramm-Elemente eine eckige Form haben, heben sich die rund eingekreisten Überschriften genügend vom Rest des Struktogramms ab, um eine Fehlinterpretation als weiteres Struktogramm-Element

zu vermeiden.

Die Struktogramme können nun so eingesetzt werden, dass ein Struktogramm immer genau einer Prozedur (bzw. booleschen Funktion oder Rekursion) entspricht, und daher auch mit dem umgangssprachlichen Synonym dieser Prozedur (bzw. des neu definierten Testbefehls oder der Rekursion) beschriftet wird.

Jedes Struktogramm ergibt sich als eine Kombination dreier graphischer Elemente (siehe Tabelle 1.)[3]

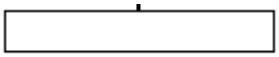
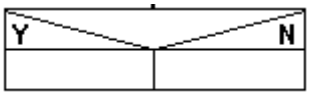
Anweisung / Prozedur	**Bedingte Verzweigung**	**Bedingte Wiederholung**
		

Tabelle 1: Elemente eines Struktogramms

4.1.2 Einsatz von Synonymen

Ein wesentlicher Vorteil der Struktogramme ist eine weitestgehende Loslösung von der konkret eingesetzten Programmiersprache. Anstatt sich beim Entwurf eines Algorithmus mit Syntax-Fragen zu beschäftigen, kann man sich auf den Programmablauf konzentrieren. Dabei ist es wichtig, klar zwischen Struktogramm und Quelltext zu unterscheiden. Eine solche Unterscheidung kann erfolgen, indem das Verwenden von Quelltext in einem Struktogramm generell untersagt wird. Stattdessen soll der Programmablauf im Struktogramm in Umgangssprache beschrieben werden.

Zu diesem Zweck bietet es sich an, für die elementaren Anweisungen der Hamstersprache umgangssprachliche Synonyme einzuführen. Durch Lernen am Modell können die

[3]Prinzipiell gibt es noch die Möglichkeit, zwischen elementaren Anweisungen und Prozeduren zu unterscheiden. Der NSD-Editor bietet für Prozeduren ein eigenes graphisches Element an (ein Rechteck mit eingeschlossener Ellipse). Dadurch könnte man den Zusammenhang zwischen der Beschriftung eines Struktogramms mit dem Namen der dargestellten Prozedur und dem Prozeduraufruf graphisch noch stärker hervorheben. Es hat sich jedoch gezeigt, dass ein solches Hervorheben für ein Verständnis der Struktogramme nicht nötig ist. Daher kann zugunsten der Einfachheit der Struktogramme und der Zeitersparnis beim Erstellen auf diese Unterscheidung verzichtet werden.

Schüler anhand der Vorgabe solcher Synonyme selbst entsprechende Formulierungen für eigene Unterprozeduren erstellen.
Umgangssprachliche Synonyme sind zum Beispiel in der Tabelle 2 abgebildet.

Quelltext	**Umgangssprachliches Synonym**
linksUm();	Drehe Dich nach links.
vor();	Gehe einen Schritt vor.
nimm();	Nimm ein Korn auf.
gib();	Gib ein Korn ab.

Tabelle 2: Synonyme für elementare Anweisungen

Die umgangssprachlichen Synonyme für die vier elementaren Anweisungen beim Hamster sind entsprechend zweier Grundregeln gewählt:

1. Sie sind in Form von Befehlen formuliert.

2. Sind sind in Form von ganzen, vollständigen Sätzen formuliert.

Die erste Regel ergibt sich trivialerweise, da auch der Quelltext in Form von Anweisungen gegeben ist, bringt aber auch zusätzlich weitere Vorteile mit sich, wenn zum Beispiel in der Partnerarbeit, ein Programm in Form eines Struktogramms nachvollzogen werden soll, und der eine Schüler die Anweisungen gibt, während der andere Schüler mit entsprechendem Material die Anweisungen ausführt.
Die zweite Regel soll erstens verhindern, dass Mischformen von Quelltext und umgangssprachlichen Anweisungen auftreten. Zum Beispiel könnte die umgangssprachliche Anweisung 'Nimm.' schnell dazu führen, dass die Schüler entweder nicht mehr 'Nimm.', sondern 'Nimm();', 'nimm.()', 'Nimm;' usw. schreiben, was dann als unerwünschten Nebeneffekt Syntaxprobleme in der Implementierungsphase mit sich bringt, oder dass sie gar keine umgangssprachlichen Formulierungen mehr für die Struktogramme wählen, wodurch diese einen wesentlichen Nutzen, nämlich die graphische Darstellung eines Algorithmus losgelöst von der Programmiersprache, verlieren würden.

Zweitens soll so das Ziel erreicht werden, dass sich die Struktogramme wie eine Geschichte lesen und einfacher nachvollziehen lassen.
Analog zu den umgangssprachlichen Synonymen für die elementaren Anweisungen gibt es auch umgangssprachliche Synonyme für die drei möglichen Testbefehle. Dabei gibt es jedoch im Unterschied zum Quelltext für jeden Testbefehl *zwei* mögliche Formulierungen, die in Abhängigkeit vom verwendeten Kontext (bedingte Wiederholung oder bedingte Verzweigung) eingesetzt werden, um die unterschiedliche Charakteristik dieser beiden Kontrollstrukturen zu betonen.
Für die bedingten Verzweigungen bietet sich eine Formulierung in Form von Zustandsabfragen an, während sich für die bedingte Wiederholung eine Formulierung in Form einer Bedingung anbietet.

Quelltext	**Bedingte Verzweigung**	**Bedingte Wiederholung**
vornFrei()	Ist vorne frei?	Solange vorne frei ist,
kornDa()	Ist ein Korn da?	Solange ein Korn da ist,
maulLeer()	Ist das Maul leer?	Solange das Maul leer ist,

Tabelle 3: Synonyme für Testbefehle

Verneiungen, die im Quelltext mit einem vorgestellten Ausrufezeichen erzeugt werden, können im Struktogramm durch die vorgestellten Wörter 'nicht' bzw. 'kein' dargestellt werden. Die Darstellung von 'UND' und 'ODER'-Verküpfungen ergibt sich natürlich aus ihrer Bezeichnung.
Auch hier hat die klar vorgegebene Struktur eines Testbefehls in Form einer Frage bzw. einer Bedingung den Vorteil, dass die Schüler entsprechend diesen beispielhaften Vorbildern selbstdefinierte Testbefehle sinnvoll und einheitlich ausformulieren können.

4.2 Einsatzmöglichkeiten der Struktogramme

Es ergeben sich nun vielfältige Möglichkeiten, wie sich die Struktogramme didaktisch einsetzen lassen.

4.2.1 Zusammenpuzzeln

Es gibt verschiedene Möglichkeiten, wie man das Zusammenpuzzeln von Programmen aus vorgegebenen Struktogramm-Bausteinen im Unterricht einsetzen kann. Die Puzzle-Methode hat den Vorteil, dass die Schüler sich haptisch und spielerisch mit dem Erstellen von Programmen beschäftigen können.

Einführen einer Kontrollstruktur Die Methode des Zusammenpuzzelns kann dafür eingesetzt werden, dass der Aufbau einer bisher unbekannten Kontrollstruktur erarbeitet wird. So können Puzzelteile mit elementaren Anweisungen, mit Testbefehlen und leeren Schleifen an die Schüler verteilt werden, mit dem Arbeitsauftrag, diese zu Miniprogrammen zusammenzupuzzeln, um die Schüler so den Aufbau einer while-Schleife erarbeiten zu lassen.

Anwenden einer Kontrollstruktur Ist der grundsätzliche Aufbau einer Kontrollstruktur bekannt, so kann die Anwendung der Kontrollstruktur ebenfalls mit der Puzzle-Methode geübt werden, wie hier beispielhaft anhand der while-Schleife erklärt werden soll.
Üblicherweise wird die Verwendung der while-Schleife motiviert, indem der Lehrer auf den Trick zurückgreift, die Hamster-Landschaft variabel zu gestalten, so dass beispielsweise die Größe der Landschaft nicht bekannt ist. Da die Landschaft nun nicht mehr fest vorgegeben ist, kann sie nur beispielhaft als Bild angegeben werden und muss in der Regel textuell über ihre Eigenschaften beschrieben werden.
Dadurch ergeben sich für die Schüler jedoch gleichzeitig zwei neue Schwierigkeiten. Sie müssen sich erstens aus einer textuellen Beschreibung der Landschaft eine Vorstellung von dieser Landschaft schaffen und zweitens die neue Kontrollstruktur verstehen.

Außerdem sind die Landschaften zwangsläufig sehr schlicht bis langweilig gehalten. Eine Alternative dazu besteht nun darin, die Landschaft fest zu lassen, aber die Häufigkeit einzuschränken, mit der eine elementare Anweisung im Quelltext vorkommen darf. Auch dadurch wird zum Beispiel der Einsatz der while-Schleife erzwungen, jedoch ohne die zusätzliche Schwierigkeit des Verstehens einer variablen, textuell beschriebenen Landschaft.
Dieses Einschränken der Häufigkeit muss für die Schüler motiviert werden. Dies kann einerseits über Einsicht geschehen, indem der Lehrer seine Beweggründe mitteilt. Es kann jedoch zusätzlich auf einer unbewußteren Ebene geschehen, indem der Lehrer eine Arbeitssituation schafft, in der dieses Einschränken unumgänglich ist. Eine Situation, in der sich der Schüler nicht freiwillig auf die Verwendung dreier Anweisungen 'Gehe einen Schritt vor.' beschränkt, sondern nicht mehr als diese drei Anweisungen zur Verfügung hat.
Eine solche Arbeitssitutuation wird dadurch geschaffen, dass der Lehrer Struktogramm-Bausteine in Form von ausgeschnittenen Puzzlestücken vorgibt, aus denen das gewünschte Programm zusammengebaut werden soll. Im Grunde wird dadurch eine unnötige Code-Duplizierung verhindert, wodurch sich die Qualität der Software erhöht.

Binnendifferenzierung Schließlich eignet sich die Puzzle-Methode im besonderen Maße, um Arbeitsaufträge zu formulieren, die auf eine starke Binnendifferenzierung hinauslaufen. So können zum Beispiel jeweils zwei Schüler einen Satz von Puzzleteilen erhalten mit dem Auftrag, ein Programm zusammenzustellen, das etwas bestmöglich erledigt:
Üblicherweise soll ein Programm entworfen werden, das ein konkretes Problem löst (z.B.: 'Hamster läuft zur Höhle und sammelt dabei alle Körner auf.'). Entweder die Schüler scheitern und das Programm tut nicht, was es soll, oder sie sind erfolgreich, und es tut, was es soll. Die Bewertung der Lösungen läßt sich noch etwas abstufen, indem der Lehrer angibt, ob die Lösung von guter oder schlechter Qualität ist, und dies begründet. Was dabei oft fehlt, ist ein objektiver Maßstab, an dem sich die Schüler

unmittelbar orientieren können.
Wird nun jedoch eine bestmögliche Lösung verlangt, so bedeutet es, dass es keine Klassifizierung zwischen richtiger und falscher Lösung mehr gibt, so dass im günstigsten Fall jede Lösung richtig ist, aber von unterschiedlicher Qualität. Diese wiederum läßt sich objektiv daran messen, wie gut das Programm erledigt, was es erledigen soll, anstatt dass sie vom Lehrer bestimmt wird. (z.B.: 'Stelle mit den gegebenen Bausteinen ein Programm zusammen, so dass der Hamster in der dargestellten Landschaft so weit wie möglich nach rechts läuft.') Das Ziel des Schülers eine Software bestimmter Qualität zu entwickeln, tritt hier deutlich in den Vordergrund.

4.2.2 Nachvollziehen

Sollen die Schüler neue Kontrollstrukturen erlernen, so kann der Lehrer als eine erste Übung auch fertige Programme in Form von Struktogrammen und einer Hamsterlandschaft mit der Aufforderung vorgeben, das Programm nachzuvollziehen. Um sicherzustellen, dass die Schüler dies auch tatsächlich tun und um die Ergebnisse gut vergleichen zu können, kann der Lehrer den Arbeitsauftrag insofern erweitern, dass die Schüler den Weg, den der Hamster nimmt, in die Landschaft eintragen sollen. Die Lösung läßt sich anschließend gut anhand einer Musterlösung auf Folie vergleichen, indem sie entweder während der weiteren Arbeitsphase zum selbstständigen Vergleichen herumgereicht und auf die eigene Lösung draufgelegt wird, oder indem sie am OHP besprochen wird. Für eine solche Aufgabenstellung bietet es sich übrigens an, die Zeilen und Spalten der Landschaft zu nummerieren, damit man anschließend besser darüber sprechen kann.
Außerdem läßt sich bei dieser Aufgabenstellung gut eine Binnendifferenzierung zwischen stärkeren und schwächeren Schülern vornehmen, indem nicht ein Programm zum Nachvollziehen vorgegeben wird, sondern zwei mit unterschiedlicher Komplexität. Soll zum Beispiel die bedingte Wiederholung als als Kontrollstruktur geübt werden, so könnte das eine Programm ineinander geschachtelte Schleifen beinhalten,

während das andere diese noch nicht beinhaltet.

4.2.3 Vervollständigen

Als Variante zur vorherigen Vorgehensweise mit erhöhter Schwierigkeit könnte der Lehrer auch den Weg des Hamsters und einen Teil der Struktogramme vorgeben mit der Aufgabenstellung, das Programm zu vervollständigen.

4.2.4 Erstellen

Letztendliches Ziel ist es dann natürlich, dass die Schüler selbst in der Lage sind, zu einem gegebenen Problem eine Lösung in Form von Struktogrammen zu entwerfen. Beim Erstellen der Struktogramme habe ich mich dafür entschieden, dass dies in der Anfangsphase auf einem Blatt Papier zu geschehen hat. Zwar gibt es gute frei erhältliche Struktogramm-Editoren (siehe 7.2), mit denen dies auch erledigt werden kann, doch gibt es mehrere Gründe dies nicht zu tun:

1. Die deutliche Trennung zwischen Theoriephase und Rechnerarbeit wäre nicht mehr gegeben.

2. Die Versuchung für die Schüler gleichzeitig (bzw. ersatzweise) schon mal die Hamsterumgebung zu starten, um das Problem 'direkt' zu lösen, wäre wohl zu groß.

3. Das Zeichnen der Struktogramme per Hand führt zu einer deutlicheren Bewußtwerdung der graphischen Darstellung und ihrer Bedeutung. (Zum Beispiel gut zu beobachten beim Zeichnen des Strukturelements für die bedingte Verzweigung.)

4.2.5 Erfahren des Modulprinzips

Dadurch, dass ein Programm mit Unterprozeduren als Ansammlung von Struktogrammen dargestellt wird, wird sein Modulcharakter visuell hervorgehoben. Die Schüler

können unmittelbar *sehen*, dass das Programm aus mehreren Teilblöcken zusammengesetzt ist, von denen einige nur einmal durchlaufen werden während andere sehr häufig wiederverwendet werden. [4]
Diese starke Visualisierung des Modulscharakters einer Programmiersprache läßt sich nun an verschiedenen Stellen ausnutzen:

Top-Down-Prinzip Ein wichtiges Prinzip beim Programmentwurf ist die Top-Down-Vorgehensweise (nach dem 'divide et impera'-Prinzip, siehe [5], S.45).
Um nun die Top-Down-Vorgehensweise einzuführen, könnte man zum Beispiel die Bezeichner zu verwendender Struktogramme (Prozeduren) vorgeben mit der Aufgabenstellung, deren Rumpf auszuformulieren. Der Lehrer gibt also einen Hinweis, wie sich das Problem in Unterprobleme zerlegen ließe. Die Aufgabe besteht dann darin, die Schnittstellen zwischen den Unterproblemen zu überlegen und Lösungen dafür zu formulieren.
Alternativ könnte man auch *ein Struktogramm mit dem kompletten Programmablauf* vorgeben, mit der Aufgabenstellung, dieses Struktogramm so in Unterstruktogramme zu zerlegen, dass das Hauptprogramm möglichst kurz und verständlich wird.
Letztendliches Ziel ist es natürlich, dass die Schüler selbstständig das Problem in geeignete Unterprobleme zerlegen, welche sie in einzelnen Struktogrammen lösen. Da die Hamsterprogramme in der Regel recht übersichtlich sind, wird die gesamte Lösung einschließlich der Hamsterlandschaft auf eine oder zwei OHP-Folien passen, so dass sie anschließend auch sinnvoll präsentiert werden kann.

Wiederverwendbarkeit Beim Hamster ergeben sich schnell bestimmte Unterprozeduren, die in den meisten Programmen wiederverwendet werden (z.B. 'Drehe Dich nach rechts.', 'Gehe eine Stufe hinauf.', 'Drehe Dich um.' usw.) und durch die immer

[4] Zwar läßt sich dies prinzipiell auch dem Quelltext entnehmen, jedoch hat man es da in der Regel nicht so plastisch vor Augen, da sich die Unterprozeduren über mehrere Bildschirmseiten verteilen und die Übersichtlichkeit auch stark vom Vermögen der Schüler abhängt, den Quelltext durch geeignete Formatierungen sinnvoll zu gliedern.

einheitliche visuelle Darstellung in Form eines Struktogramms einen gewissen Wiedererkennungswert gewinnen. Sie werden auch visuell als häufig wiederverwendetes Modul erkannt.
An dieser Stelle könnte der Lehrer zum Beispiel Programme in Struktogrammform vorgeben, mit der Aufgabe zu analysieren, wie häufig bestimmte Module verwendet werden. Basierend auf den Ergebnissen dieser Analyse könnten die Ursachen erforscht werden (Warum wird das Modul xyz eigentlich so oft verwendet?) und anschließend Schlüsse gezogen werden, ob es sich um einen sinnvoll realisierten Algorithmus handelt. So ist es scheinbar unsinnig, andere Prozeduren als die main-Prozedur zu erstellen, die nur ein einziges Mal ausgeführt werden. Jedoch können auch solche Prozeduren Sinn machen, wenn sie die Lesbarkeit des Programms wesentlich verbessern.
Dadurch verschiebt sich auch ein wenig die Schwerpunktsetzung im Unterricht. Anstatt dass die Schüler einfach eine irgendwie geartete Lösung für ein Problem finden sollen (was eventuell sogar sehr einfach ist), sollen sie sich Gedanken zur Qualität einer Lösung machen bzw. eine qualitativ hochwertige Lösung erstellen.
Die Darstellung über eine kompakte Ansammlung von Struktogrammen ermöglicht auch einen unmittelbaren visuellen Vergleich verschiedener Lösungen eines Problems: Angenommen es gibt zwei verschiedene Programme zur Lösung eines Problems, die aber gemeinsame Unterprozeduren verwenden. Dann lassen sich die beiden Programme als Mengen von Struktogrammen darstellen, in deren Schnittmenge die gemeinsam genutzten Struktogramme liegen.

Softwareentwicklung als Teamwork Nach Einführung der Kontrollstrukturen kann sich der Lehrer etwas größere Problemstellungen überlegen, die sich im Team entsprechend der Top-Down-Vorgehensweise in Teilprobleme zerlegen lassen. Im Team muss dann geklärt werden, was die Lösungen der Teilprobleme im Einzelnen zu leisten haben, und wie die Schnittstellen aussehen müssen. Die Schnittstellen sind in diesem Fall die Start- und die Endsituation des Hamsters vor bzw. nach Bearbeitung eines Teilproblems.

Die Teilprobleme werden dann anschließend auf die einzelnen Gruppenmitgleider verteilt und werden einzeln in Form von Struktogrammen gelöst. Anschließend werden alle Struktogramme gesammelt und zu einem Gesamtprogramm zusammengelegt, welches dann gemeinsam auf Korrektheit überprüft werden kann. Hierbei kann erstens die Kompaktheit der Struktogramme und zweitens ihr Vorteil beim Nachvollziehen fremder Lösungen ausgenutzt werden.
Werden die einzelnen Struktogramme außerdem jeweils auf einen Folienschnipsel geschrieben, so kann das Gesamtprogramm anschließend auch auf dem Overheadprojektor zusammengelegt und der Klasse vorgestellt werden.
Es ist an dieser Stelle übrigens wichtig, den Schülern zu vermitteln, dass hier im Kleinen nachgestellt wird, was im Großen die Regel ist. Ansonsten könnte es bei der üblicherweise eher geringen Komplexität der Hamsterprogramme nicht wirklich einsichtig sein, wieso nicht jeder Schüler für sich alle Programmteile erstellen soll.

5 Analyse und Evaluation des Konzepts

Das im vorherigen Kapitel beschriebene Konzept wurde von mir in einer Jahrgangsstufe 11 am Gymnasium getestet und bewertet. Dabei ist anzumerken, dass mir einige der genannten Ideen zu den Einsatzmöglichkeiten der Struktogramme erst später gekommen sind. Der Unterrichtsreihe zugrunde gelegen hat das Buch 'Programmieren spielend gelernt' von D.Boles ([2]) und sein Vorgängerbuch ([1]). Eine Fortsetzung der Reihe wäre mit dem Band 2 zur objektorientierten Hamsterprogrammierung ([3]) denkbar, wurde aber in diesem Fall nicht gemacht.
In der Unterrichtsreihe wurde Wert darauf gelegt, den gesamten Prozess der Softwareentwicklung im Kleinen nachzubilden, wozu insbesondere das Kapitel 3 aus [2] als Hilfe herangezogen wurde.
Die Rechnerarbeit wurde in der beschriebenden Art und Weise organisiert, die Struktogramme von den Schülern per Hand auf einem Blatt Papier angefertigt. Es wurde versucht, die jeweiligen Hamsteraufgaben in kleinere Geschichten einzubetten, um ihnen zumindest innerhalb der Hamsterwelt eine gewisse Sinnhaftigkeit zu geben, und die Aufgaben so etwas interessanter zu gestalten.
Eine genauere Beschreibung der Reihe würde den Rahmen dieser Arbeit sprengen, die ja in erster Linie das zugrundeliegende Konzept beschreiben soll. Um dennoch einen detaillierteren Eindruck denkbarer Aufgabenstellungen zu ermöglichen, sind dem Anhang exemplarisch einige Arbeitsmaterialien hinzugefügt.

5.1 Ziel der durchgeführten Unterrichtsreihe

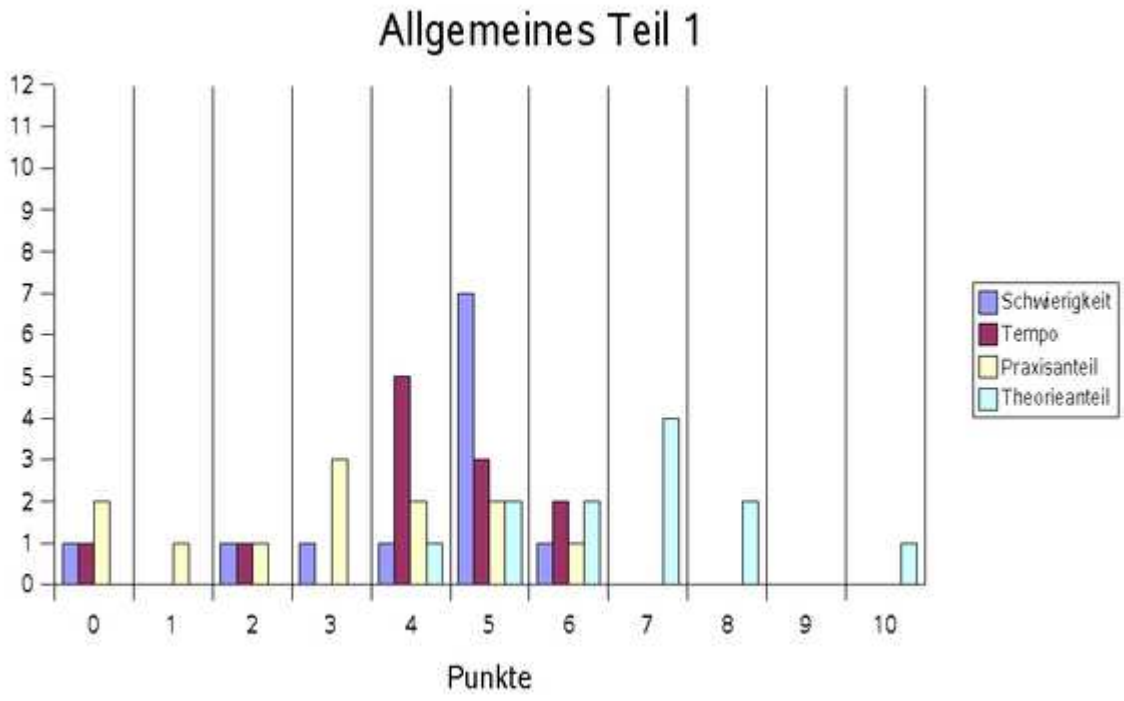
Die Schüler sollten am Ende der Reihe selbstständig imperative Programme zur Lösung eines Problems in der Java-Hamster-Sprache entwickeln können, indem sie zunächst unter Beachtung wichtiger Prinzipien der Lösungsfindung einen (weitestgehend) programmiersprachenunabhängigen Lösungsentwurf unter Verwendung von Struktogrammen als Werkzeug zur Visualisierung imperativer Algorithmen erstellen, und diesen erst in einem zweiten Schritt implementieren.

Dazu erlernten sie im Verlauf der Reihe unter anderem den Aufbau und die Funktionsweise grundlegender Kontrollstrukturen einer Programmiersprache.

5.2 Auswertung einer abschließenden Umfrage

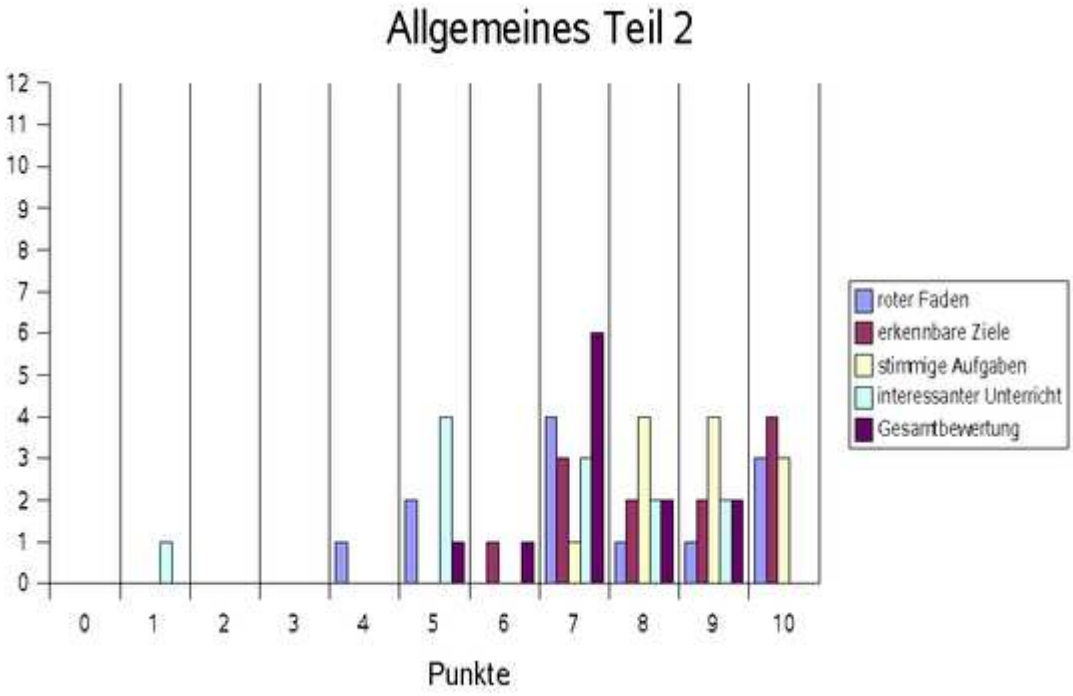
Am Ende der Unterrichtsreihe wurde von mir in der Klasse eine anonyme Umfrage gemacht, mit der ich erfahren wollte, welche Einstellung die Schüler gegenüber der Verwendung von Struktogrammen haben.
Nun kann aber eine positive bzw. negative Grundhaltung gegenüber dem Unterricht im Allgemeinen bzw. der Verwendung des Hamsters im Speziellen die Frage nach dem Nutzen von Struktogrammen verfälschen. Daher habe ich den Schülern mehrere entsprechende Fragen gestellt, die scheinbar nichts mit Struktogrammen zu tun haben, aber deren Antwort die Antwort auf die Frage beeinflussen können, ob Struktogramme hilfreich sind.
Dadurch ergibt sich eine Abhängigkeit zwischen den einzelnen Antworten eines Schülers, so dass zu beachten ist, dass eine graphische Repräsentation der Daten der ganzen Klasse zu einer einzelnen Frage diese Abhängigkeit nicht ohne weiteres berücksichtigen kann. Zur graphischen Darstellung der Daten habe ich mich für ein Histogramm entschieden, da dadurch erstens verhältnismäßig wenig Information verloren geht, und man zweitens gut Tendenzen der gesamten Klasse erkennen kann.
Außerdem wurden auch redundante Fragen gestellt, um Hinweise auf emotional beeinflusste Antworten zu erhalten. Beispielsweise gibt es eine Frage, ob der Anteil an Praxis und eine danach, ob der Anteil an Theorie angemessen war. Andere Fragen sollten es ermöglichen einen besseren Gesamteindruck von der Meinung bzw. Einstellung des einzelnen Schülers zu erhalten (siehe 7.5).
Die Klasse umfaßt insgesamt 12 Schüler und es haben auch alle Schüler an der Umfrage teilgenommen. Die Umfrage gliederte sich in zwei Teile. Im ersten Teil wurden neun Einschätzungen zum Unterricht im Allgemeinen und im zweiten Teil acht Einschätzungen zum Inhalt des Unterrichts erfragt. Die Einschätzungen sollten ge-

nerell auf einer Skala von 0 bis 10 Punkten erfolgen. Bei den ersten vier Fragen des ersten Teils liegt das Optimum in der Mitte der Skala. (siehe erstes Diagramm) Bei den restlichen Fragen repräsentiert die 10 das Optimum.
Die Ergebnisse der Umfrage werden im Folgenden in drei verschiedenen Histogrammen dargestellt. Auf der Abzisse sind die Punkte der Einschätzungsskala abgetragen und auf der Ordinate ist die Anzahl der Schüler angegeben, die auf eine Frage mit einer bestimmten Punktzahl geantwortet haben.
Es ist zu beachten, dass es sich aufgrund der Daten um eine diskrete Darstellung handelt. Daher kann ein Balken immer nur eine ganzzahlige Höhe haben, die in einigen Fällen auch Null beträgt, weshalb es keinen Balken gibt.

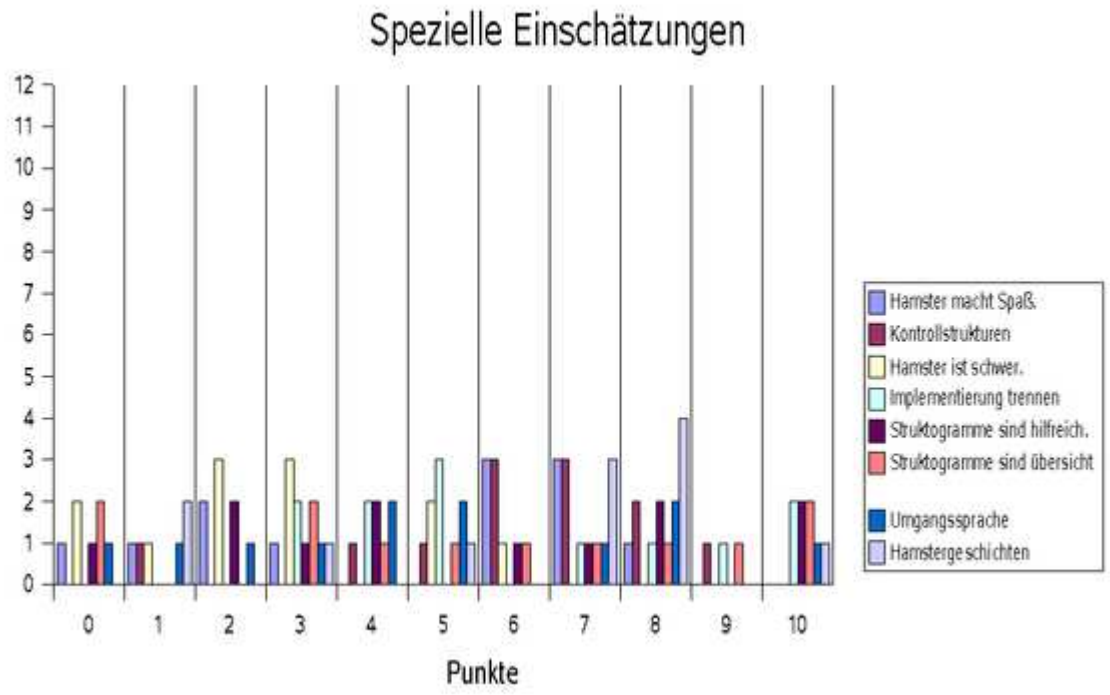


Interpretation: Dem Histogramm läßt sich entnehmen, dass die Schüler den Theorieanteil im Unterricht tendenziell als etwas hoch und den Praxisanteil tendenziell als etwas niedrig empfunden haben. Das Tempo wurde im Allgemeinen als angemessen eingeschätzt mit leichter Tendenz dazu, dass es etwas langsam war. Die Schwierigkeit

wurde als angemessen empfunden.



Interpretation: Zunächst einmal fällt auf, dass bis auf zwei Ausreißer alle Einschätzungen in der oberen Hälfte (beginnend bei 5 Punkten) liegen. Der Unterricht wurde unterschiedlich interessant empfunden – es gibt eine breite Streuung von *ok* bis *gut*. Die Unterrichtsziele und der rote Faden waren *ganz gut* bis *sehr gut* zu erkennen. Auffallend ist die sehr positive Einschätzung von Stimmigkeit der Aufgaben zur Theorie. Insgesamt wird der Unterricht mehrheitlich als *ganz gut* (7 Punkte) eingeschätzt.



Interpretation: Zunächst einmal scheint es keine so deutlichen Tendenzen wie zuvor zu geben. Bei genauerem Hinsehen fällt Folgendes auf:

1. Die Programmierung mit dem Hamster wird tendenziell als leicht eingestuft.
2. Die Programmierung mit dem Hamster wird von den Schülern tenenziell als geeignet empfunden, um damit neue Kontrollstrukturen zu erlernen.
3. Die Planung eines Programms von seiner Implementierung zu trennen wird im Schnitt als mittelmäßig wichtig bis wichtig eingestuft, variiert jedoch stark von Schüler zu Schüler.
4. Der Einsatz von Hamstergeschichten wurde von der Hälfte der Klasse positiv bewertet. Zur Nützlichkeit der Struktogramme scheint es keinen Konsens in der Klasse zu geben. Die Ergebnisse variieren stark von absoluten Verfechtern bis absoluten Gegnern der Struktogramme. Der Durchschnitt beider Fragen zu den Struktogrammen liegt nur leicht oberhalb der Mitte.

5.3 Betroffene Lehrerfunktionen

5.3.1 Unterrichten

Der größte Einfluß auf das Unterrichten durch den Einsatz von Struktogrammen erfolgt insofern, dass eine klare Trennung zwischen Rechnerarbeitsphase und Programmplanungsphase ermöglicht wird. Dadurch wird die Rechnerarbeitszeit effektiver und zielgerichteter genutzt.

5.3.2 Diagnostizieren und Fördern

Bei der Softwareentwicklung treten in der Regel zwei verschiedene Typen von Fehlern auf. Einerseits die Syntaxfehler, die zu Fehlermeldungen zur Kompilierzeit führen, und andererseits die Semantikfehler, die zur Laufzeit auftreten.
Wird ein Programm direkt am PC entwickelt, so kommt es oft vor, dass Syntaxfehler die semantischen Fehler verdecken, d.h. es gibt bereits semantische Fehler im Programm, die aber noch nicht bemerkt werden, da das Programm aufgrund von Syntaxfehlern noch nicht kompiliert wurde, und die semantischen Fehler sich erst zur Laufzeit bemerkbar machen.
Im ungünstigsten Fall ist es vielleicht sogar so, dass der Teil mit dem Syntaxfehler auch einen semantischen Fehler beinhaltet. Das könnte zur Folge haben, dass der Syntaxfehler behoben wird, nur um kurz darauf feststellen zu müssen, dass dieser Teil des Quelltextes so nicht bestehen bleiben kann.
Daher sollte die Behebung von Fehlern soweit wie möglich in der Reihenfolge erfolgen, dass erst die Semantikfehler behoben werden und dann die Syntaxfehler. Dies geht aber nur dann, wenn beim Programmentwurf eine Trennung zwischen Semantik und Syntax ermöglicht wird. Der Einsatz von Struktogrammen bietet eine solche Möglichkeit. Erst wird ein semantisch korrektes Programm in Form von Struktogrammen entworfen, welches anschließend syntaktisch korrekt implementiert werden muss. Der angenehme Nebeneffekt davon ist, dass sich auch die Diagnose und Förderung von seitens des Lehrers durch diese Trennung vereinfacht.

Erstens kann er erkennen, welcher Teil der Programmentwicklung einem Schüler mehr Probleme bereitet und entsprechend darauf reagieren. Hat ein Schüler in der ersten Phase Probleme, so muss er etwa noch lernen, wie man prinzipiell einen Algorithmus entwirft oder verstehen, wie die neue Kontrollstruktur funktioniert.
Hat er Probleme während der Rechnerarbeitsphase, so liegt die Ursache höchstwahrscheinlich im Umgang mit der Entwicklungsumgebung, dem Erkennen verschiedener Syntaxfehler, der Nutzung von Fehlermeldungen und so weiter.
Zweitens kann der Lehrer selbst die vorhandenen Programmfehler schneller erkennen und entweder entsprechend gezieltere Hinweise an den Schüler geben, wie dieser den Fehler entdecken kann, oder ihm den Fehler nennen, wenn der Schüler weder alleine noch mit Hilfe der Mitschüler weiterkommt.

5.3.3 Beraten und Erziehen

Einerseits wird die inhaltliche Beratung durch den Lehrer vereinfacht, wie bereits im Abschnitt zur 'Diagnose und Förderung' erläutert wird. Andererseits werden die Schüler dazu erzogen, dass sie im weiteren Unterricht in der Teamarbeit berücksichtigen, dass sie genaue Schnittstellen definieren müssen, um eine erfolgreiche Gruppenlösung zu erstellen, dass sie erst gründlich eine Lösung planen, bevor sie sie tatsächlich implementieren, und dass sie sich nach geeigneten Visualisierungen für ihre Lösungen umsehen, um sie verständlich präsentieren zu können.
Nach der Einführung der wichtigsten Kontrollstrukturen einer prozeduralen Programmiersprache ist oft das nächste Thema die objektorientierte Softwareentwicklung. In diesem Zusammenhang bekommen die obigen Aspekte eine neue Bedeutung:

Schnittstellen Welche Attribute und Methoden sollen die Klassen haben? Was sollen diese Methoden leisten?

Trennung von Entwurf und Implementierung Erst werden die Klassendiagramme erstellt, dann werden die Klassen implementiert.

Präsentieren mit geeigneter Visualisierung Klassendiagramme in UML-Notation werden als Visualisierungshilfe genommen

Man sieht, dass sich durch geschickten Einsatz der Struktogramme beispielhaft viele fundamentale Ideen in der Informatik vermitteln lassen, die anschließend bei der objektorientierten Softwareentwicklung völlig analog wieder aufgegriffen und damit auch gefestigt werden können.

5.3.4 Leistung messen und beurteilen

Der konsequente Einsatz der Struktogramme als pädagogisches Werkzeug führt unter anderem zu einer sehr klaren Trennung der Unterrichtsphasen. Entwurf, Präsentation und Implementierung eines Algorithmus können in drei aufeinander folgende Phasen erfolgen. In der ersten Phase wird der Algorithmus in Form eines Struktogramms entwickelt, in der zweiten Phase wird der Algorithmus mit Hilfe des Struktogramms den Mitschülern präsentiert und in der dritten Phase wird das Struktogramm als Leitfaden für die Implementierung eingesetzt.

Dadurch wird es dem Lehrer ermöglicht, die Leistungen der Schüler deutlich bestimmten Kompetenzen zuordnen zu können. Hat ein Schüler in der ersten Phase Probleme, so liegt seine Schwäche sehr wahrscheinlich im Erstellen eines Algorithmus (vorausgesetzt natürlich, dass es nicht noch Verständnisprobleme bei den Struktogrammen gibt, was jedoch aufgrund ihrer Einfachheit nach einiger Übung eher unwahrscheinlich ist).

Hat er jedoch erst Probleme in der zweiten Phase, so kann er sich nicht klar genug ausdrücken, hat Hemmungen vor der Klasse zu stehen oder dergleichen.

Gibt es erst in der dritten Phase Probleme, so hat der Schüler grundsätzlich ein Verständnis für den Algorithmus und kann ihn auch erklären, aber er hat Probleme mit der Entwicklungsumgebung, der Syntax der Programmiersprache oder beim Umgang mit dem Computer.

Möchte man einen Eindruck davon bekommen, dass dies nicht unbedingt selbst-

verständlich ist, falls man den Unterricht anders organisiert, sei auf ein fiktives, aber realistisches Fallbeispiel im Anhang verwiesen.

5.3.5 Organisieren und Verwalten

Ein wesentlicher Vorteil der Struktogramme besteht darin, dass sie ermöglichen, den Anteil echter Lernzeit an der gesamten Unterrichtszeit zu erhöhen. Dies geschieht dadurch, dass ihr Einsatz eine klare Trennung und möglichst wenig Wechsel zwischen Rechnerarbeitsphasen und anderen Arbeitsphasen impliziert.
Außerdem ergibt sich für den Lehrer eine Arbeitserleichterung bei der Bewertung der erstellten Programme. Statt alle Programme einsammeln zu müssen, um anhand des Quelltextes herauszufinden, was für einen Algorithmus der Schüler erstellt hat, kann er die Struktogramme einsammeln oder während der Planungsphase beim Herumgehen begutachten. Da er nur auf die Semantik achten muss, besteht eine gute Chance, dass er tatsächlich einen angemessenen Eindruck vom individuellen Fortschritt der Schüler bekommt. Gleiches gilt für die Rechnerarbeitsphase, in der der Lehrer idealerweise hauptsächlich auf die Syntax der Programme achten muss, bzw. ob sie am Ende funktionieren.

5.3.6 Evaluieren, Innovieren und Kooperieren

Die Innovation dieses Konzepts besteht darin, Struktogramme nicht nur als eine mögliche Visualisierung eines Algorithmus vorzustellen, sondern sie gezielt als technisches, didaktisches und erzieherisches Hilfsmittel im Informatikunterricht einzusetzen, um damit fundamentale Ideen der Informatik gegenüber nicht allgemeinbildenden Inhalten in den Vordergrund zu stellen und die Rechnerarbeit sinnvoll zu organisieren.

6 Fazit

Gemäß den Umfrageergebnissen in der Klasse gibt es weder eine klare Befürwortung des Einsatzes der Struktogramme durch eine Mehrheit der Klasse noch eine klare Ablehnung.
Als eine wesentliche Ursache dafür vermute ich, dass das *Zeichnen* der Struktogramme den Nachteil hat, dass Fehler nachträglich nur schlecht korrigiert werden können. Daher kann das Zeichnen der Struktogramme zu einem Unmut führen, der eigentlich nichts mit den Struktogrammen selbst sondern mit dem damit verbundenen Zeichnen zu tun hat.
Die Schüler haben es gemäß der Umfrage nämlich durchaus als wichtig eingeschätzt, die Planung eines Programms von seiner Implementierung zu trennen. Dies mag nun entweder an der Einsicht liegen, dass dies wichtig ist, oder daran, dass das vorherige Planen der Programme mit den Struktogrammen den Schülern beim Lösen der Aufgaben geholfen hat. Ich vermute eher die zweite Ursache, was mich in der beschriebenen Nutzung der Struktogramme bestärkt.
Die spezielle Idee, Programme aus Struktogrammelementen zusammenpuzzeln zu lassen, kam mir erst nach der Unterrichtsreihe in der 11. Ich vermute, dass diese spielerische Methode, die Akzeptanz der Struktogramme wesentlich erhöhen würde, da hier keine Notwendigkeit besteht, Struktogramme zeichnen zu müssen.
Insgesamt hat sich die Vorgehensweise, die Struktogramme als ein Werkzeug einzusetzen, um die Schüler zu einer der Implementierungsphase vorangehenden theoretischen Planungsphase zu erziehen, sehr gut bewährt. Diese theoretische Planungsphase hat es auch ermöglicht, einige etwas gößere Hamsteraufgaben in kleinen Gruppen zu bearbeiten, was am Rechner aufgrund der durch die Hamster-Entwicklungsumgebung gesetzten Grenzen im Team gar nicht sinnvoll möglich gewesen wäre. Schließlich gestalteten sich die Präsentationen der Schüler durch die Struktogramme sehr strukturiert und verständlich.

7 Anhang

7.1 Literaturverzeichnis

Literatur

[1] Programmieren spielend gelernt mit dem Java-Hamster-Modell, Dietrich Boles, ISBN: 3-8351-0064-5, Teubner

[2] Der Hamster – Programmieren lernen in einer Modellwelt, Wolfgang Ambros, ISBN: 3-476-20378-6, J.B.Metzlersche Verlagsbuchhandlung Stuttgart

[3] Objektorientierte Programmierung spielend gelernt mit dem Java-Hamster-Modell, Dietrich Boles, Cornelia Boles, ISBN: 3-519-00506-9, Teubner

[4] Ikarus – Natur und Technik, Schwerpunkt: Informatik 6/7, Brichzin Freiberger, Reinold Wiedemann, ISBN: 3-486-88286-4, Oldenburg

[5] Abenteuer Informatik – IT zum Anfassen von Routenplaner bis Online-Banking, Jens Gallenbacher, ISBN: 3-8274-1635-3, Elsevier - Spektrum Akademischer Verlag

[6] Informatik 1 – Objekte, Strukturen, Algorithmen, Elke Frey, Peter Hubwieser, Ferdinand Winhard, ISBN: 3-12-731468-x, Klett

[7] Grundlagen der Programmierung, Das Einsteigerseminar, Oliver Müller, bhv, ISBN: 3-8287-1024-7

[8] Computerzeitung, Die Wochenzeitung für das IT-Management, 38.Jahrgang Nr.18 von 2007

[9] Pädagogisches Grundwissen, Herbert Gudjons, ISBN: 3-7815-0888-9, Klinkhardt

[10] Pädagogik, Herausgeber: Hermann Hobmair, ISBN: 3-8237-5000-3, Bildungsverlag EINS

[11] Unterrichtsempfehlungen für den Wahlpflichtunterricht für die Gesamtschule in Nordrhein-Westfalen - Informatik, Herausgeber: Kultusministerium des Landes Nordrhein-Westfalen, ISBN: 3-89314-336-X, Verlagsgesellschaft Ritterbach mbh

[12] Sekundarstufe II Gymnasium/Gesamtschule - Richtlinien und Lehrpläne, Herausgeber: Ministerium für Schule und Weiterbildung, Wissenschaft und Forschung des Landes Nordrhein-Westfalen, ISBN: 3-89314-612-1, Ritterbach Verlag GmbH

[13] Didaktik der Informatik: Grundlagen, Konzepte, Beispiele, Peter Hubwieser, Berlin 2000, ISBN 3-540-65564-6

7.2 Internetlinks

- Die Hamster-Homepage vom Erfinder des Hamsters Dietrich Boles:
  http://www.java-hamster-modell.de/
- Wikipedia zu Struktogrammen:
  http://de.wikipedia.org/wiki/Struktogramm
- NSD-Struktogramm-Editor:
  http://diuf.unifr.ch/softeng/student-projects/completed/kalt/NSD.html
- StructEd-Struktogramm-Editor:
  http://www.strukted.de/
- StgrWin32-Struktogramm-Editor:
  http://www.uni-koblenz.de/ ceinig/dl/index.php?filecat=1
- Vips-Struktogramm-Editor:
  http://www.partheil.com/vips/index.html

7.3 Fiktives Fallbeispiel zu 'Leistung messen und beurteilen'

Angenommen, die Klasse bekommt die Aufgabe, in Einzelarbeit ein Hamsterprogramm zu schreiben. Das Programm darf direkt am Rechner erstellt werden. Während der Rechnerarbeit bemerkt der Lehrer beim Herumgehen, dass ein Schüler scheinbar größere Schwierigkeiten hat und nicht so schnell vorankommt, wie seine Mitschüler. Am Ende der Arbeitsphase hat er aber trotzdem ein sehr gutes funktionierendes Hamsterprogramm. Als er sein Programm präsentieren soll, geschieht dies nur sehr schleppend und unsicher.
Der Lehrer möchte die Schülerergebnisse gerne bewerten und ist ratlos, was er sich zu der Leistung des Schülers notieren soll. Denn:

1. Hatte der Schüler während der Arbeitsphase Probleme beim Entwickeln eines Algorithmus oder bei seiner Implementierung?

2. Ist die Lösung überhaupt vom Schüler selbst, wenn er sie nur dermaßen schlecht vorstellen kann, oder liegt es an der mangelnden Kompetenz des Schülers etwas zu präsentieren? Oder könnte er den Algorithmus eigentlich sogar ganz gut präsentieren, aber er kommt durch die Unübersichtlichkeit des Quelltextes durcheinander?

3. Der Lehrer ahnt nun, dass ihm ein bestimmter Mitschüler geholfen hat. Aber hat dieser ihm bei Syntaxproblemen oder beim Entwurf des Algorithmus geholfen?

7.4 Erklärungen

7.4.1 Urheberrecht

Ich versichere, dass ich die Arbeit eigenständig verfasst, keine andere Quellen und Hilfsmittel als die angegebenen benutzt und die Stellen der Arbeit, die anderen Werken dem Wortlaut oder Sinn nach entnommen sind, in jedem einzelnen Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe. Das Gleiche gilt auch für beigegebene Zeichnungen, Kartenskizzen und Darstellungen.

_____

Sebastian Funke

7.4.2 Einverständniserklärung

Ich bin damit einverstanden, dass diese Hausarbeit nach Abschluss meiner Zweiten Staatsprüfung wissenschaftlich und pädagogisch interessierten Personen oder Institutionen zur Einsichtnahme zur Verfügung gestellt wird und dass zu diesem Zweck Ablichtungen dieser Hausarbeit hergestellt werden, sofern diese keine Korrektur- oder Bewertungsvermerke enthalten.

_____

Sebastian Funke

7.5 Umfrage und Arbeitsmaterial

Die folgenden Seiten beinhalten den zweiseitigen Fragebogen der Umfrage, die am Ende der Unterrichtsreihe durchgeführt wurde, sowie exemplarisch einige Arbeitsmaterialien, die verdeutlichen sollen, welche verschiedenen Varianten es gibt, mit Java-Hamster-Struktogrammen zu arbeiten.
Die Bilder in den Arbeitsblättern sind Screenshots der Hamster-Entwicklungsumgebung.
Die Arbeitsblätter sind so angeordnet, wie sie nacheinander in einer Unterrichtsreihe zum Hamster eingesetzt werden können, um die folgenden Inhalte in der angegebenen Reihenfolge zu behandeln:

1. bedingte Wiederholung
2. bedingte Verzweigung
3. Der Programmentwurf
4. boolesche Funktionen
5. Rekursionen

Das Arbeitsblatt zur Hamsterin (drittes Blatt) zeigt die Verwendung eines anderen Layouts für die Struktogramme, welches zu Irritationen führt, da der äußere Rahmen als Struktogramm-Element angesehen werden kann.
Die Ideen, den Hamster nicht nur aus der Vogelperspektive von oben sondern alternativ von der Seite zu sehen, und ihn über einen Berg laufen zu lassen und die Slalomaufgabe habe ich von meinem Ausbildungslehrer übernommen. Die restlichen Aufgaben sind in der Regel von mir selbst erstellt und ansonsten dem Hamsterbuch ([2]) entnommen.