Informatik für Alle –
Wie viel Programmierung braucht der Mensch?

Dissertation

zur Erlangung des mathematisch-naturwissenschaftlichen Doktorgrades

"Doctor rerum naturalium"

an der Georg-August-Universität Göttingen

vorgelegt von

Kerstin Maike Strecker

aus Kiel

Göttingen 2009

Mitglieder des Betreuungsausschusses:

Prof. Dr. D. Hogrefe (Referent)

Prof. Dr. W. Herget (Koreferent)

Prof. Dr. E. Modrow

Tag der mündlichen Prüfung: 30.10.2009

Mein Dank gilt Prof. Dr. Hogrefe, Prof. Dr. Herget und insbesondere Prof. Dr. Modrow für die Betreuung dieser Arbeit.

Inhaltsverzeichnis

1 Einleitung....3
2 Zur Definition von Programmieren – Systemkonfiguration als didaktische Reduktion....7
2.1 Was ist Programmieren?....7
2.2 Codierung....7
2.3 Analyse und Design....9
2.4 Objektorientierte Modellierung – objects first....11
2.5 Systemkonfiguration....14
2.5.1 Imperative Programme....16
2.5.2 Systemkonfiguration und Algorithmik....19
2.5.2.1 Algorithmik als fundamentale Idee der Informatik....19
2.5.2.2 Systemkonfiguration in Abgrenzung zur Algorithmik....21
2.6 Systemkonfiguration als didaktischer Rahmen im Anfangsunterricht....26
2.7 Systemkonfiguration und Modellierung....28
2.8 Zusammenfassung....30
3 Inhaltliche Ausgestaltung des Programmierens in der Sekundarstufe I....32
3.1 Programmierunterricht und Weltorientierung....32
3.2 Beschreibung eines Techniksystems....33
3.3 Konfigurationen in einem Techniksystem....34
3.4 Algorithmenbausteine zur Konfiguration technischer Systeme....37
3.4.1 Steuerungssysteme....40
3.4.2 Formalisierung....45
3.5 Konfigurationssprachen technischer Systeme....46
3.5.1 Graphische oder textbasierte Programmierung?....46
3.5.2 Verschiedene „Programmierparadigmen“ in der Konfiguration technischer Systeme....52
3.5.2.1 Systemkonfiguration und funktionales Paradigma....53
3.5.2.2 Systemkonfiguration und imperatives Paradigma....59
3.6 Struktogramme vs. UML-Aktivitätsdiagramme....63
3.7 Zusammenfassung....66
4 Zur Methodik einer Unterrichtseinheit „Programmieren für Alle“....68
4.1 Methodik zum Erarbeiten der Systemkonfigurationen....68
4.1.1 Dekonstruktion von Systemen....68
4.1.2 Erkenntnisse aus der Mathematikdidaktik....69
4.1.3 Umgang mit Fehlern....76

4.2 Methodische Überlegungen bzgl. der Unterrichtseinheit „Programmieren“ ... 77
4.2.1 Zweckorientierter Unterricht mit konkreten eigenen Produkten ... 77
4.2.2 Fächerübergreifender Unterricht ... 80
4.2.3 Projektunterricht ... 81
4.3 Zusammenfassung ... 82
5 Die PuMa-Lernumgebung – Hausautomation im Puppenhaus ... 84
5.1 Haussteuerungssysteme ... 84
5.2 Das PuMa-System ... 85
5.3 Die Evaluation ... 91
5.4 PuMa und das LEGO-System ... 94
5.5 Die Beschreibung der Konfigurationen in PuMa ... 96
5.5.1 Das Puma-System und „scratch“ ... 98
5.5.2 Konfigurationsumgebungen in PuMa, die noch zu realisieren sind ... 100
5.6 Kategorisierung der Aufgaben im PuMa-System ... 102
5.7 Zusammenfassung ... 106
6 Unterrichtssequenzen ... 108
6.1 Konfigurieren in verschiedenen Lernumgebungen mit einer anschließenden Systematisierung ... 109
6.1.1 Die Unterrichtssequenz im Schuljahr 2007/2008 ... 109
6.1.1.1 Übersicht des Unterrichtsverlaufs ... 109
6.1.1.2 Die Programmierumgebungen ... 110
6.1.1.3 Programmiereinheiten: Durchführung und Ergebnisse ... 111
6.1.1.4 Fazit ... 118
6.1.1.5 Evaluation ... 119
6.2 Programmieren in einer Lernumgebung und verschiedenen Oberflächen mit einer anschließenden Systematisierung ... 122
6.3 Übergang zu textbasierten Sprachen am Beispiel „scratch“ und Delphi – eine Unterrichtssequenz im Schuljahr 2008/2009 ... 123
6.4 Zusammenfassung ... 128
7 Zusammenfassung der Arbeit ... 129
Abbildungsverzeichnis ... 133
Tabellenverzeichnis ... 136
Literaturverzeichnis ... 137

1 Einleitung

„Informatik für Alle – Wie viel Programmierung braucht der Mensch?“ Unter dieser etwas plakativ formulierten Fragestellung sind Überlegungen zusammengefasst, die sich auf inhaltliche, methodische und didaktische Aspekte des Informatikunterrichts und speziell des Themenbereiches „Programmieren“ allgemeinbildender Schulen, hier: des Gymnasiums, beziehen. Informatikunterricht „für Alle“ meint, insbesondere Schülerinnen und Schüler in den Fokus zu nehmen, die Informatik *nicht* als Schwerpunktfach im Abitur wählen und eine berufliche Richtung in diesem Gebiet einschlagen. Vielmehr müssen Schülerinnen und Schüler im Mittelpunkt der Überlegungen stehen, die keine besondere technische, mathematische oder informatische Begabung aufweisen. Um alle Schülerinnen und Schüler erreichen zu können, konzentriert sich diese Arbeit auf Konzepte des Programmierunterrichtes der Sekundarstufe I[1]. Ein Mittelstufenunterricht, der lediglich die Inhalte der Sekundarstufe II vorbereitet, reicht nicht aus. Wenn aber Lernende im Mittelpunkt der Betrachtungen stehen, die Informatik *nicht* als Fach in der Oberstufe fortführen, dann reicht dies natürlich erst recht nicht aus. Es müssen eigene, in sich geschlossene Konzepte vorliegen, die die Frage beantworten können, warum auch diese Schülerinnen und Schüler programmieren sollten. Ein Unterricht im „Programmieren“ kann nur dann sinnvoll sein, wenn Kompetenzen vermittelt werden können, die in anderen Schulfächern oder Unterrichtseinheiten in dieser Form nicht vermittelbar sind und gleichzeitig einen allgemeinbildenden Wert darstellen, der den Lernenden hilft, die sie umgebende Lebenswelt besser zu verstehen und einordnen zu können.

Die didaktischen, inhaltlichen und methodischen Konzepte eines Informatikunterrichts in der Sekundarstufe I werden in dieser Arbeit auf den Themenbereich „Programmierung“ begrenzt. Eine Notwendigkeit von „Programmieren“ im Informatikunterricht wird in der Didaktik jedoch kontrovers diskutiert. Peter Hubwieser resümiert: „Bei näherer Betrachtung haben solche Differenzen oft ihre Ursache in einer unterschiedlichen Interpretation des Begriffs „Programmierung““ (Hubwieser 2004). Daher ist zunächst eine intensive Erläuterung dieses Begriffs notwendig, auf deren Fundament sich die anschließende Argumentation dieser Arbeit aufbaut. Beim Versuch einer solchen Definition in Kapitel 2 wird die Vielschichtigkeit der Anforderungen im Programmieren an die Lernenden sichtbar, weshalb eine didaktische Reduktion

[1] Mit der Einführung des achtjährigen gymnasialen Ausbildungsganges (G8) in allen Bundesländern ändert sich in mehreren Ländern die Gliederung in:
Sekundarstufe I: Klassen 5 bis 9
Sekundarstufe II: Klassen 10 bis 12
Nach überwiegender Nomenklatur gliedert sich die Sekundarstufe I in die Unterstufe (Klasse 5 und 6) und die Mittelstufe (Klasse 7 bis 9). Mit Oberstufe wird die Sekundarstufe II bezeichnet (wikipedia 2009d).

vorgenommen werden muss. Es kristallisieren sich verschiedene Handlungsschemata in der Programmierung heraus. Neben der Anforderung der Codierung, also der Übersetzung eines Algorithmus in eine bestimmte Sprache, wurde in verschiedenen didaktischen Arbeiten das Augenmerk auf eine (objektorientierte) Modellierung und den Entwurf von Systemen gelegt, (siehe dazu u. a. Brinda 2004). Diese Arbeit beschreitet nun einen anderen Weg. Mathematisch gesehen kann ein Programm als Transformation von Eingabedaten in Ausgabedaten verstanden werden. Verwendet der Lehrende einerseits Lernumgebungen in einem System, welches intuitiv zu erfassen ist, also keine komplexen Strukturen aufweist, die eine Modellierung nötig machen, und wählt der Lehrende andererseits eine Programmiersprache[2], in der Schülerinnen und Schülern keine Syntaxfehler unterlaufen können, dann ist die Programmieranforderung die übrig bleibt, eine Konfiguration dieses Systems für die Erfüllung eines bestimmten Zwecks. Diese *Systemkonfiguration* wird als Transformation von Eingaben in Ausgaben *algorithmisch* beschrieben, wie ausführlich in Kapitel 2 dargelegt werden wird. Wir reduzieren die Anforderung der Programmierung damit didaktisch auf die Konstruktion einer Funktion, die aus einigen elementaren algorithmischen Grundbausteinen aufgebaut wird und ein bestehendes System so konfiguriert, dass ein vorher bekannter und erwünschter Endzustand erreicht wird. Dabei werden bei der Wahl der Grundbausteine weitere Einschränkungen gemacht, die zu didaktischen und methodischen Vorteilen führen und in Kapitel 2 näher erläutert werden. Insbesondere sind Konfigurationen ihrerseits aus beliebig kombinierbaren Funktionen aufgebaut, die einen kompositionellen Entwurf nahe legen und immer zu funktionsfähigen Systemen führen.
Keineswegs bewegt sich diese Arbeit dabei zur Unterrichtskultur der 80er Jahre zurück, in denen Programmierunterricht aus dem Anfertigen sequentieller, imperativer Programme z. B. in TurboPascal oder Basic (siehe u. a. Prante 1978; Hermes 1990; Baumann 1992; Griesel 1992) bestand. Vielmehr zeigt diese Arbeit, dass unterschiedliche Konfigurationen der verschiedensten Systeme in den unterschiedlichsten Sprachen quer durch alle Gebiete der Informatik auf den unterschiedlichsten Niveaus das algorithmische Grundverständnis der Lernenden verbessern (siehe dazu auch Kapitel 6)[3].

Auch im didaktischen Rahmen der Systemkonfiguration müssen die Lerngegenstände dem Allgemeinbildungsanspruch genügen, der bei einem Unterricht „für Alle“ die zentrale Rolle spielt.

[2] Der Begriff einer Programmiersprache ist in dieser Arbeit sehr weit gefasst, nähere Erläuterungen finden sich in Kapitel 2.

[3] Eine Modellierung oder ein Systementwurf kann sich sinnvoll an eine Systemkonfiguration anschließen. Sind die Schülerinnen und Schüler in der Lage, bekannte Systeme gemäß eigenen Anforderungen zu konfigurieren, dann können im nächsten Schritt eigene neue Systeme entworfen werden, sofern die Lernenden aus entwicklungs-psychologischer Sicht zu den notwendigen Abstraktionsleistungen fähig sind.

Informatik ist am Gymnasium eines der wenigen Fächer, meist das einzige, mit technischem Bezug und kann *inhaltlich* dazu beitragen, dass Schülerinnen und Schüler die sie umgebende technische Welt besser durchdringen. Unter dem Anspruch der „Weltorientierung“ (siehe Heymann 1996) beschränkt sich diese Arbeit in Kapitel 3 auf die Konfiguration *technischer* Systeme. Sie zeigt, wie es Schülerinnen und Schülern dadurch gelingt, ein Verständnis für die algorithmische Funktionsweise technischer Systeme zu gewinnen und dies auf Alltagsgeräte zu transferieren.

Technische Systeme in verschiedenen Umgebungen und Sprachen zu konfigurieren, in welchen Sensoren Eingabewerte liefern und Aktoren durch eine geeignete Kombination von Verarbeitungsschritten gesteuert werden, ermöglicht das Erkennen aller algorithmischen Grundbausteine. Dabei liefern technische Systeme zusätzlich nötige Beschränkungen in der Komplexität und Struktur der Algorithmen, die der Autorin für einen Programmierunterricht der Sekundarstufe I in Abgrenzung zu Inhalten der Oberstufe angemessen erscheinen. Im Hinblick auf die Nebenläufigkeit in technischen Prozessen erweitern sie bestehende Vorgehensweisen im Unterricht sogar (siehe dazu u. a. Frey, Hubwieser, Winhard 2004; Engelmann 2008).

Die methodischen Wege zu dem Ziel, Lernende zu befähigen, die Funktionsweise technischer Systeme ihrer Lebensumwelt algorithmisch zu beschreiben, wird in Kapitel 6 dargestellt. Diese Arbeit zeigt, wie Lernende aufgrund unterschiedlichster Erfahrungen in der Konfiguration in den verschiedensten Kontexten lernen, diese sinnvoll zu systematisieren und auf unbekannte Systeme zu transferieren. Allerdings müssen die gemachten Erfahrungen der Schülerinnen und Schüler bestimmten Anforderungen genügen. Aus Erkenntnissen der Mathematikdidaktik (siehe u. a. Bruder, Leuders, Büchter 2008) leitet diese Arbeit methodische Vorgehensweisen bei der Programmierung ab. Entgegen bekannten Ansätzen (Hampel, Magenheim, Schulte 1999) der „Dekonstruktion von Systemen“ mit dem Ansatz: „... anstelle der Neuimplementation der eigenen Modellierung dekonstruieren die Lernenden eine Implementation, die auf den Analyse-, Design- und Implementationsentscheidungen eines anderen „Entwicklungsteams“ beruht“ (Hampel, Magenheim, Schulte 1999), zeigt die Mathematikdidaktik, dass gerade die *eigenständige* Algorithmensuche der Schülerinnen und Schüler, also die *selbstständige* Suche einer Transformation, die Eingabezustände in Endzustände transformiert, Kompetenzen fördert, die im bisherigen Unterricht unterrepräsentiert sind, und damit die Kompetenzen der Lernenden um einen entscheidenden Aspekt erweitert. Dies ist auch deshalb möglich, weil die Beschränkung auf technische Systeme (Kapitel 3) Vereinfachungen mit sich bringt und die Beschreibungsmittel der Konfigurationen (siehe Kapitel 2) vielfältige Lösungszugänge ermöglichen.

Dieser oben aufgezeigte didaktische, inhaltliche und methodische Rahmen einer Unterrichtseinheit „Programmieren für Alle“ wird in den Kapiteln 2 bis 4 dieser Arbeit genauer ausgeführt. Mit dem dort entwickelten Anforderungskatalog, wird eine neuartige Lernumgebung begründet und vorgestellt. Kapitel 5 beschreibt das von der Autorin entwickelte PuMa-System (Hausautomation im Puppenhaus) als Beispiel eines *realen*, technischen, auf die Lebenswelt der Lernenden übertragbaren Systems und zeigt Ergebnisse des Unterrichts.

Der Bericht und die Evaluation zweier Unterrichtssequenzen bilden Kapitel 6. In einem Fall werden geeignete Lernumgebungen verwendet, die auf dem Sensor-Aktor-Prinzip beruhen und verschiedensten Programmierparadigmen gerecht werden. Die Schülerinnen und Schüler sammeln Erfahrungen in der eigenständigen Konfiguration dieser Systeme und strukturieren und systematisieren ihr Wissen anschließend, so dass sie befähigt werden, auch Funktionalitäten technischer Systeme algorithmisch zu beschreiben, die nicht Unterrichtsgegenstand sind. Die zweite Unterrichtseinheit beschreibt den Einsatz des PuMa-Systems mit verschiedenen Konfigurationsumgebungen, deren gewählte Syntax aus unterschiedlichen Bereichen der Informatik stammt. Gleichzeitig wird die Komplexität der Programmieraufgaben (zusammen mit den Überlegungen aus Kapitel 3) klassifiziert, und es werden geeignete Aufgabenkategorien für den Unterricht in der Sekundarstufe I begründet.

Kapitel 7 fasst schließlich die Ergebnisse dieser Arbeit zusammen und gibt einen Ausblick auf offene Fragestellungen.

2 Zur Definition von Programmieren – Systemkonfiguration als didaktische Reduktion

2.1 Was ist Programmieren?

Wie bereits in der Einleitung erwähnt, besteht eine kontroverse Diskussion einiger Didaktiker und Fachwissenschaftler über eine Notwendigkeit von Programmieren im Informatikunterricht. Hubwieser (Hubwieser 2004) führt das auf eine unterschiedliche Interpretation des Begriffs „Programmierung“ zurück.

Unstrittig wird die Formulierung des Informatik-Handbuchs sein, wo Pomberger definiert: „*Programmieren im Sinne der Informatik heißt, ein* ***Lösungsverfahren*** *für eine* ***Aufgabe*** *so zu* ***formulieren****, dass es von einem Computer ausgeführt werden kann*“ (Pomberger, Rechenberg 2002).

Man kann in dieser Definition verschiedene Dinge unterschiedlich stark betonen:

1. Stehen im Vordergrund die *Aufgabe* und deren Analyse im Hinblick auf die Suche nach ihrer Lösung?
2. Stehen im Vordergrund das *Lösungsverfahren* und die Mittel, die zur Konstruktion der Lösung zur Verfügung stehen?
3. Steht im Vordergrund die *Formulierung* der Lösung, so dass ein Computer sie ausführen kann, also die Codierung in einer Programmiersprache?

Bei der Tätigkeit des Programmierens sind also viele verschiedene Kompetenzen und Aspekte zu berücksichtigen. Je nachdem, auf welche der drei genannten Anforderungen ein Schwerpunkt gelegt wird, gibt es Argumente, die Programmieren im Unterricht mehr oder eben weniger sinnvoll erscheinen lassen. Im Folgenden werden die drei Anforderungen detaillierter betrachtet.

2.2 Codierung

Nach Meinung der Autoren des von der Gesellschaft für Informatik e.V. herausgegebenen Vorschlags für „Bildungsstandards Informatik für die Sekundarstufe I“ (Puhlmann et al. 2008) ist die Formulierung der Lösung, die Codierung, ebenfalls Lerngegenstand im Unterricht:

„Schülerinnen und Schüler der Jahrgangsstufe 8 bis 10 lesen formale Darstellungen von Algorithmen und setzen sie in Programme um“ (Puhlmann et al. 2008).

Nach Ansicht der Autorin darf der reinen Codierung im Unterricht kein zu großes Gewicht verliehen werden. Der Codegenerator „Innovator“ (MID 2009) beispielsweise erzeugt automatisch C#-Code aus UML-Diagrammen. Ginge es also nur um die reine Übersetzung eines fertigen Algorithmus in eine Programmiersprache, würde das keine große kognitive Leistung der Schülerinnen und Schüler erfordern. Mangelt es der Codierung als ausschließlichem Lerngegenstand damit an allgemeinbildendem Wert, so ist sie dennoch unumgänglich, wenn die Algorithmen von Computern ausgeführt werden sollen. Erfahrungen in der Schule zeigen, dass ein Großteil der Zeit bei der Codierung in textbasierten Sprachen dafür aufgewendet werden muss, Syntaxfehler zu finden und zu korrigieren, während die Erstellung oder das Verständnis für den zugrunde liegenden Algorithmus bereits verinnerlicht ist. Da Zeit in der Schule aber leider immer begrenzt ist, stellt sich die Frage, wie man die Codierung zeitlich angemessen begrenzen kann. Graphische Programmierumgebungen, die Syntaxfehler ausschließen, bieten hier eine Lösung und werden in Kapitel 3 näher diskutiert. An dieser Stelle sollen Zitate von Schülerinnen und Schülern einer 11. Klasse in einer freien schriftlichen Evaluation deutlich machen, dass es einer reinen Codierung nicht nur an allgemeinbildendem Lernzuwachs mangelt, sondern dass dies eine Tätigkeit ist, die die Lernenden im Allgemeinen ziemlich frustriert. (Anmerkung: Automaten-Kara ist eine graphische Programmierumgebung, die in folgenden Kapiteln näher beschrieben wird, siehe dazu Reichert, Nievergelt, Hartmann 2005).

Abbildung 1: Schülermeinung zum Unterschied zwischen Automaten-Kara und Java

„Karas Problem / Aufgabe man auch lösen kann, wenn man irgendwo ein Komma oder „;“ Semikolon weniger hat.“

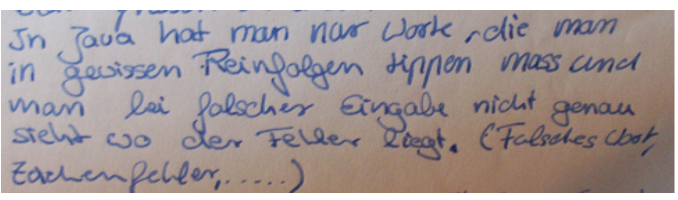

In Java hat man nur Worte, die man in gewissen Reinfolgen tippen muss und man bei falscher Eingabe nicht genau sieht wo der Fehler liegt. (Falsches Wort, Zeichenfehler, ...)

Abbildung 2: Schülermeinung zum Unterschied zwischen Automaten-Kara und Java

„In Java hat man nur Worte, die man in gewissen Reihenfolgen tippen muss und man bei falscher Eingabe nicht genau sieht, wo der Fehler liegt. (Falsches Wort, Zeichenfehler, ...)“

Sieht man im Programmieren also nur die reine Codierung, dann wäre nach Meinung der Autorin eine Unterrichtseinheit „Programmieren für Alle“ nicht zu legitimieren.

2.3 Analyse und Design

Wenn die Codierung außer acht gelassen wird, dann bleiben aus obiger Definition folgende Punkte bestehen:

1. Stehen im Vordergrund die *Aufgabe* und deren Analyse im Hinblick auf die Suche nach ihrer Lösung?
2. Stehen im Vordergrund das *Lösungsverfahren* und die Mittel, die zur Konstruktion der Lösung zur Verfügung stehen?

Versucht man diese beiden Punkte gegeneinander abzugrenzen, dann geht es im ersten Punkt zunächst einmal um eine Analysetätigkeit. Die Aufgabe und damit das System, in dem diese Aufgabe platziert ist, muss analysiert und modelliert werden.

Beispiel 1:

Eine *Aufgabe* an Lernende der Sekundarstufe II, vielleicht im Rahmen eines Schulprojekts, könnte lauten: *Programmiert für die Lehrerinnen und Lehrer die Vorlage eines elektronischen Zeugnisses, in das die Noten eingegeben werden und welches Schülerinnen und Schüler sich online ansehen können.*

Bevor die Lernenden jetzt eine Oberfläche generieren und ihr Programm schreiben, müssen Fragen geklärt werden:

- Wer ist an der Erstellung eines Zeugnisses im Normalfall beteiligt? (Lehrende?, Schulleitung?)
- Wer vergibt die Noten im Arbeits- und Sozialverhalten? Wird darüber abgestimmt?
- Was passiert in der Zeugniskonferenz ? Ist das wichtig für unser Problem?
- Wie verhindert man, dass nicht nur die Empfänger des Zeugnisses ihre Noten online sehen?
- Wie können Eltern und die Schulleitung ein elektronisches Zeugnis unterschreiben?

Mit anderen Worten: hier ist eine Analysetätigkeit erforderlich. Das System „elektronisches Zeugnis“, seine Akteure und Systemgrenzen sowie die Interaktionen im System müssen zunächst einmal analysiert und geeignet modelliert werden. Zur Zeit bedient man sich in Praxis und Unterricht dabei der Modelle und Diagramme der UML (Unified Modeling Language) (siehe dazu Oesterreich 2006; Oose 2001; Schmuller 2000).
Erst wenn in der Analysephase eindeutig spezifiziert ist, was genau implementiert werden soll, kann sich die Konstruktion des Lösungsverfahrens und damit Punkt 2 anschließen. Für den zweiten Punkt muss das System, in dem diese Aufgabe zu lösen ist, bekannt sein. Ein zu erreichender Endzustand oder eine Ausgabe des Systems müssen präzise definiert sein.
Die Konstruktion des Lösungsverfahrens kann als Programmdesign verstanden werden. Es ist inzwischen in Wissenschaft und Praxis der Informatik deutlich gemacht worden, dass sich ein striktes Trennen von Analyse und Design nicht bewährt hat und auch gar nicht möglich ist. Deshalb treten in der heutigen Softwareentwicklung auch nur noch spiralförmige Vorgehensweisen oder Vorgehensweisen wie das OEP (Object Engineering Process) (Oesterreich 2006) auf und nicht etwa das sogenannte „Wasserfallmodell“ (Floyd, Züllighoven 2002). Trotz einer nicht immer durchführbaren Trennung zwischen den Punkten 1 und 2 gibt es dennoch einige charakteristische Unterschiede.
Je nachdem, welcher der Punkte 1 oder 2 unter dem Aspekt „Programmierung“ stärker betont wird, werden unterschiedliche Kompetenzen von den Schülerinnen und Schülern gefordert und gefördert.
Wird der Schwerpunkt im Anforderungsfeld der Programmierung auf den ersten Punkt gelegt, dann werden von den Lernenden insbesondere prozessbezogene Kompetenzen der *Modellierung* gefordert. Schubert und Schwill beispielsweise sehen durch die Modellierung einer Lösung mittels einer Analyse der Aufgabenstellung und des zugrunde liegenden Systems Grundkompetenzen gefördert, die sie für wichtig halten (Schubert, Schwill 2004).

In den letzten Jahren ist die Modellierung im Informatikunterricht sehr in den Vordergrund der Diskussionen gerückt, auch mit der Einführung des objektorientierten Paradigmas und der objektorientierten Programmiersprachen an den Schulen (siehe dazu Hubwieser 2004; Brinda 2004; Schubert, Schwill 2004). „Hubwieser entwickelte ein geschlossenes Konzept zum informatischen Modellieren für den Informatikunterricht, das sich an den *Phasen des Software-Entwicklungsprozesses orientiert, wobei der Schwerpunkt auf der Modellierungsphase* und dem Erlernen und Anwenden von Modellierungstechniken liegt“ (Brinda 2004). Unter dem Schlagwort „objects first“ zeigt auch die Arbeit von Ira Diethelm (Diethelm 2007) Beispiele für eine objektorientierte Modellierung in der Schule.
Der Weg, der in den letzten Jahren in der Informatikdidaktik damit beschritten worden ist, legt also den Schwerpunkt in der Anforderung an die Programmierung auf Punkt 1 oder die (objektorientierte) Modellierung. Der nächste Abschnitt beschäftigt sich mit diesem Ansatz im Informatikunterricht.

2.4 Objektorientierte Modellierung – objects first

Das objektorientierte Paradigma eröffnet neue Möglichkeiten der Analyse von Aufgaben und steht eng im Zusammenhang mit Modellierungstechniken der UML. Wie sehr sich dieser Zugang im Informatikunterricht allgemeinbildender Schulen bereits durchgesetzt hat, sollen die folgenden vier Beispiele zeigen. Beispiel 2, 3 und 4 zeigen, wie sehr dieses Konzept in die Richtlinien der Unterrichtsinhalte eingegangen ist. Beispiel 5 zeigt eine Aufgabe aus diesem Themenkomplex im Unterricht.

Beispiel 2:

Die Oberstufeninhalte (Jahrgang 10 bis 12) in Informatik sind durch die bundeseinheitlichen Prüfungsanforderungen in der Abiturprüfung (EPA) weitgehend festgelegt (siehe dazu EPA 2004). Näher spezifiziert werden diese von der Kultusministerkonferenz (KMK) festgelegten Inhalte durch die jeweiligen Themenschwerpunkte des Zentralabiturs (siehe dazu nibis 2009).
Am Beispiel des niedersächsischen Abiturs 2009 werden in Stichpunkten die verbindlichen Inhalte und Themenschwerpunkte in einem Oberstufenkurs in Informatik verdeutlicht. Die Bedeutung objektorientierter Programmiersprachen und der Lerngegenstand der objektorientierten Modellierung wird in Tabelle 1 gezeigt.

Thematischer Schwerpunkt 1: Werkzeuge und Methoden der Informatik
Algorithmen (allgemein)
- Erstellung eines Algorithmus zu einem gegebenen Problem in schriftlich verbalisierter Form oder als Struktogramm
- […] Implementierung eines Algorithmus in Java oder Pascal / Delphi [...]

Datenstrukturen und abstrakte Datentypen
[…]
- Implementierung eines neuen abstrakten Datentyps unter Abwägung verschiedener Alternativen
- Nutzung und Implementierung abstrakter Datentypen (lineare Listen, binäre Bäume)

Ergänzung für Kurse auf erhöhtem Anforderungsniveau: Objektorientierte Modellierung (mit UML)
Analyse eines vorgegebenen Klassendiagramms
- Erweiterung eines vorgegebenen Klassendiagramms
- Erstellung eines Klassendiagramms für ein vorgegebenes System
- Implementierung eines Modells unter Berücksichtigung der Konzepte der Kapselung, derVererbung und der Polymorphie in Java oder Pascal / Delphi

[...]

Tabelle 1: Auszug aus den Themenschwerpunkten im Zentralabitur 2009 Informatik in Niedersachsen (nibis 2009)

Aber auch in der Sekundarstufe I haben die Ideen der Objektorientierten Modellierung Eingang gefunden. Hubwieser zieht die Objektorientierung auch für den Anfangsunterricht in Jahrgang 6 heran als „Grundstein für den Aufbau angemessener mentaler Modelle und die Verwendung einer sauberen, ausdrucksstarken Terminologie in den späteren Jahren“ (Hubwieser 2000). „Hubwieser betont, dass es […] darum gehe objektorientiert zu analysieren und zu modellieren, keinesfalls aber zu programmieren“ (Brinda 2004).

Beispiel 3:

Versucht man, die Inhalte des Informatikunterrichts der Sekundarstufe I in den einzelnen Bundesländern zu kategorisieren, zeigt sich ebenfalls eine Tendenz zur objektorientierten Modellierung. Weeger beschreibt in seiner Arbeit (Weeger 2007) die Positionierung des Informatikunterrichts in den einzelnen Bundesländern. Aus den Daten des Jahres 2006/7 wurden Aspekte, die den Unterricht in der Sekundarstufe I an Gymnasien betreffen, bezüglich der Unterrichtseinheit „Programmieren“ herausgegriffen. Es ergibt sich folgendes Bild: In den Bundesländern, in denen der Themenkomplex „Algorithmik, Problemlösen, Programmierung“ detaillierter erläutert ist, überwiegt die Tendenz zu objektorientierter Programmierung, mit den zugehörigen Konzepten der Modellierung. Speziell in Bayern gibt es sehr detaillierte Vorgaben, wie Tabelle 2 zeigt.

Jahrgang 6:	„Grundbegriffe der objektorientierten Beschreibung von Informatiksystemen [...]“
Jahrgang 7:	„Beschreibung von Abläufen durch Algorithmen […]“
Jahrgang 9:	„[...] modellhafte Erfassung von Zusammenhängen“ insbesondere: „Begriff der Funktion, Darstellung der Struktur von Klassen und deren Beziehungen mit Hilfe von Klassendiagrammen [...]“
Jahrgang 10:	„In Jahrgangsstufe 10 beschäftigen sich die Schüler intensiver mit dem Thema Objekte und Abläufe [...],“ insbesondere „Zuständen von Objekten und algorithmische Beschreibungen von Abläufen […]“, „Beziehung zwischen Objekten, Generalisierung und Spezialisierung durch Ober- bzw. Unterklassen und in diesem Zusammenhang das Prinzip der Vererbung [...]“

Tabelle 2: Auszüge aus der inhaltlichen Struktur des Informatikunterrichts der Sekundarstufe I in Bayern (aus Weeger 2007)

Beispiel 4:

Mit dem Ziel einer *länderübergreifenden* Vereinheitlichung hat die Gesellschaft für Informatik e.V. (GI) 2008 einen Vorschlag für Bildungsstandards Informatik in der Sekundarstufe I veröffentlicht (Puhlmann et al. 2008). Zum Inhaltsbereich „Algorithmen“ ist zu lesen: „Schülerinnen und Schüler aller Jahrgangsstufen kennen Algorithmen zum Lösen von Aufgaben und Problemen aus verschiedenen Anwendungsgebieten und lesen und interpretieren gegebene Algorithmen“ (Puhlmann et al. 2008). An späterer Stelle wird noch hinzugefügt: Sie „entwerfen, implementieren und beurteilen Algorithmen“, wobei sie „algorithmische Grundbausteine zur Darstellung von Handlungsvorschriften benutzen“. Die Autoren schreiben weiter: „Im Kern sollte erreicht werden, dass Schülerinnen und Schüler sowohl diesen Gesamtprozess des Problemlösens von der Problemstellung über ein *Modell*, einen Algorithmus, das Programm bis zur Interpretation der Ergebnisse an Beispielen kennen lernen“ (Puhlmann et al. 2008).

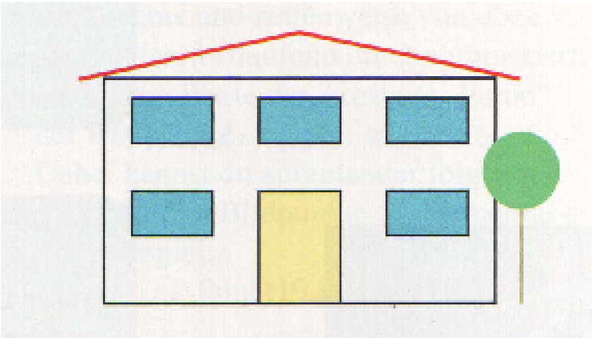
Damit wird ausgesagt, dass bei einer länderübergreifenden Vereinheitlichung der Mittelstufen-informatik im Punkt „Programmierung“ auch die Idee der objektorientierten Modellierung einge-schlossen werden soll.

Beispiel 5:

Die Schülerinnen und Schüler der Sekundarstufe I sollen in dieser Aufgabe Objekte identifizieren und sie später Klassen zuordnen (siehe dazu Abbildung 3). Ergebnisse sind Aussagen wie:

- „Das Haus hat fünf Fenster, die Objekte der Klasse Rechteck sind.“
- „Die Tür ist ein Objekt der Klasse Rechteck.“

- „Der Baum besteht aus einem Objekt der Klasse Kreis und einem Objekt der Klasse Strich.“
- „Baumkrone ist ein Objekt der Klasse Kreis. Baumkrone.Farbe=grün.“



Maxi und Karla haben zusammen ein Haus gezeichnet.

a) Zeichne das Haus. Trage für alle vorkommenden Objekte passende Bezeichner ein.

b) Welchen Klassen können die einzelnen Objekte zugeordnet werden? Gib für mindestens drei Objekte Bezeichner und Werte von wichtigen Attributen in Kurzschreibweise an.

Abbildung 3: Aufgabe aus einem Schulbuch der Sekundarstufe I (Frey, Hubwieser, Winhard 2004)

Diese Beispiele zeigen, dass die entwickelten Konzepte für den Informatikunterricht der Sekundarstufe I und dort speziell der Unterrichtseinheit Programmieren einen Schwerpunkt auf die (objektorientierte) Modellierung legen.

Diese Ansätze zeigen, dass die Schülerinnen und Schüler auch der Sekundarstufe I im Anfangsunterricht zunächst objektorientiert modellieren sollen, *bevor* sie in späteren Jahren eigene Programme schreiben. Diese Arbeit grenzt sich von dieser Vorgehensweise ab, wie im Verlauf deutlich werden wird.

2.5 Systemkonfiguration

Was bleibt, wenn das System, in dem die Aufgabe gestellt wurde, analysiert und modelliert ist? Oder wenn das System intuitiv zu erfassen ist und keiner Analyse bedarf? Wenn die Aufgabe so präzise und eindeutig spezifiziert ist, dass nur noch die Lösung konstruiert werden muss? Dann

muss das System oder dessen Modellierung, in der diese Aufgabe platziert ist, so *konfiguriert* werden, dass der gewünschte Endzustand erreicht wird.

Beispiel 6:
Eine Systemkonfiguration bedeutet zunächst einmal, ein vorhandenes System zu spezialisieren. Man spezialisiert das System Computer, um die Primzahlen zu berechnen. Man spezialisiert einen Roboter, einer schwarzen Linie zu folgen, und ein System aus Schaltern und Lampen, nur dann zu leuchten, wenn genau zwei der Schalter angeschaltet sind.

An dieser Stelle wird der Begriff Systemkonfiguration in einer ersten Annäherung folgendermaßen definiert:

> **Definition 1:** *Ist ein System gegeben, welches intuitiv zu erfassen oder ausreichend analysiert ist, dann bedeutet Systemkonfiguration, die Funktionalität dieses Systems so zu verändern, dass das System für eine bestimmte Aufgabe spezialisiert wird.*

In dieser Arbeit soll gezeigt werden, wie der didaktische Rahmen oder das didaktisch-methodische Handlungsschema *Systemkonfiguration* einen zur Modellierung alternativen ersten Zugang zur Programmierung für Schülerinnen und Schüler der Sekundarstufe I bildet.

Damit wird der Schwerpunkt der Programmierung auf Punkt 2 verlagert:

2 Stehen im Vordergrund das *Lösungsverfahren* und die Mittel, die zur Konstruktion der Lösung zur Verfügung stehen?

Im Weiteren müssen also die Mittel betrachtet werden, die zur Konstruktion einer Systemkonfiguration zur Verfügung stehen.

In der aktuellen Softwareentwicklung außerhalb der Schule sowie auch im Programmierunterricht der Sekundarstufe II stehen objektorientierte Programmiersprachen mit imperativem Kern im Mittelpunkt der Betrachtungen. So wird im Zentralabitur in Niedersachsen (nibis 2009) die Verwendung von Java oder Delphi gefordert. Das Ergebnis einer objektorientierten Modellierungsphase und eines Systementwurfs ist unter anderem eine Klassenstruktur mit verschiedenen Klassen und deren Methoden, die untereinander in Beziehung stehen. Eine Systemkonfiguration bedeutet

nun, die vorliegenden Methoden inhaltlich zu füllen, und zwar so, dass das System entsprechend den Anforderungen spezialisiert wird.

Die Implementierung einer Methode oder Angabe des entsprechenden Algorithmus folgt auch in den meisten objektorientierten Sprachen immer noch dem imperativem Paradigma. So werden auch in Delphi oder Java die Methodenimplementationen ähnlich wie in Pascal oder C imperativ angegeben, unter Nutzung der Bausteine Variablenzuweisung, Alternative, Sequenz, Schleife und anderen. In einem ersten Schritt entsprechen Konfigurationen dann den (imperativen) Methodenimplementationen. Um die *Mittel zur Konstruktion von Systemkonfigurationen* herauszuarbeiten, wird deshalb im folgenden Abschnitt zunächst versucht, Konfigurationen oder imperative Programme in mathematischen Objekten zu beschreiben.

2.5.1 Imperative Programme

Einen Programmablauf eines imperativen Programms kann man als Folge von Zustandsübergängen angeben, wobei ein Zustand $s \in Z$ eine Funktion darstellt, die jeder Variablen des Programms einen Wert der zugehörigen Wertemenge (des Typs) zuordnet. Mit Z bezeichnen wir die Menge aller Zustände.[4]

Beispiel 7:

Gegeben sei das Java-Programmstück:

```
char c;
int i;
boolean b;
```

Dann ist

$$s = \begin{pmatrix} c & i & b \\ 'a' & 5 & true \end{pmatrix} \quad \text{entspricht} \quad s = \begin{cases} c \to 'a' \\ i \to 5 \\ b \to true \end{cases}$$

ein Zustand.

Insbesondere führen wir den Zustand $s = \bot$ ein, der bei Nichtterminierung des Programms erreicht wird. Mit $Z_\bot$ wird die disjunkte Vereinigung $Z \cup \{\bot\}$ bezeichnet.

Ein sequentielles Programm sei gemäß (Best 1995) ein Wort der von der syntaktischen Variablen *SEQPROG* erzeugten Sprache. Es besteht aus einer (eventuell leeren) Reihe von Deklarationen *DECL* und einer (nicht leeren) Reihe von Kommandos *CMD*. *EXPR* bezeichnet – je nach

[4] Die Systematisierungen in Kapitel 2.5.1 gehen auf Überlegungen von Eike Best (Best 1995) zurück.

Verwendungskontext – einen arithmetischen oder booleschen Ausdruck; im Kontext *V:=EXPR* müssen die Typen von *V* und von *EXPR* übereinstimmen.

Im Folgenden bezeichne β einen Ausdruck, der boolesch sein muss.

$$SEQPROG ::= DECL; CMD \mid CMD$$

$$DECL ::= \text{var } V : SET \mid DECL_1; DECL_2$$

$$CMD ::= skip \mid abort \mid V := EXPR \mid CMD_1; CMD_2 \mid$$
$$if\ \beta_1 \rightarrow CMD_1 \lozenge \dots \lozenge \beta_m \rightarrow CMD_m\ fi \mid do\ \beta \rightarrow CMD_0\ od$$

Kontextbedingungen:

(i) Jede in *CMD* frei vorkommende Variable muss in *DECL* deklariert sein.

(ii) In *DECL* darf keine Variable doppelt deklariert sein.

(iii) Im Kontext *V:=EXPR* muss die Auswertung von *EXPR* stets einen Wert im *V* zugeordneten Wertebereich *SET* ergeben.

(iv) Die Alternative lässt keinen Nichtdeterminismus zu.[5]

Eine Funktion $f : Z \rightarrow Z_\perp$ beschreibt jetzt die in *CMD* angegebenen Kommandos formal:

(i) $f(skip, s) = s$

(ii) $f(abort, s) = \perp$

(iii) $f(V := EXPR, s) = \begin{cases} f(x) & falls\ x \neq V \\ value(EXPR) & falls\ x = V \end{cases}$

(iv) $f(c_1; c_2, s) = \begin{cases} f(c_2(f(c_1, s)) & falls\ f(c_1, s) \neq \perp \\ \perp & sonst \end{cases}$

(v) $sei\ IF \equiv if\ \beta_1 \rightarrow CMD_1 \lozenge \dots \lozenge \beta_m \rightarrow CMD_m\ fi$

$$f(IF, s) = \begin{cases} f(c_j, s) & falls\ \exists\ j \in \{1...m\} \text{ mit } value(\beta_j, s) = true \\ \perp & falls\ \ \forall\ j \in \{1..m\} \text{ gilt } value(\beta_j, s) = false \end{cases}$$

[5] Würden auch nichtdeterministische sequentielle Programme Gegenstand unserer Betrachtungen sein, müsste die Funktion f von der Menge der Zustände in die Potenzmenge der Zustände definiert sein auf Basis der Egli-Milner-Potenzmenge (Best, Strecker 2009). Da Nichtdeterminismus in der Schule aber keine Rolle spielt, vertiefen wir die Betrachtungen an dieser Stelle auch nicht weiter.

(vi)

weiterhin sei eine gültige Folge definiert als endliche Folge von Zuständen $s_0,...,s_r$ und

$\forall j,\ 0 \le j < r : value(\beta, s_j) = true \wedge s_{j+1} = f(c_0, s_j)$

$$f(DO, s_0) = \begin{cases} s_r & falls\ \exists \text{ gültige Folge } s_0,...,s_r \wedge value(\beta, s_r) = false \\ \bot & falls\ \exists \text{ gültige Folge } s_0,...,s_r \wedge s_r = \bot \\ \bot & falls\ \exists \text{ unendliche gültige Folge } s_0,..... \end{cases}$$

Die Syntax der sequentiellen imperativen Programmiersprache kann ohne Einschränkungen um ein Kommando *V:=?* erweitert werden.

Es ergibt sich je nach Wert des ? eine bestimmte Funktion:

$$f(V := ?, s) = \exists v \in IN : \begin{cases} f(x) & falls\ x \neq V \\ v & falls\ x = V \end{cases}$$

Beispiel 8:

Gegeben sei das Programmstück:

```
x=x+y;
y=x-y;
x=x-y;
```

Dieser Kommandoteil beschreibt eine Funktion, die Werte der Variablen *x* und *y* vertauscht:

$f(Beispiel, s) = f(x = x - y, f(y = x - y, f(x = x + y, s)))$ nach obiger Definition (iv).

Sei nun $s = \begin{pmatrix} x & y \\ 5 & 27 \end{pmatrix}$

Dann gilt: $f(x = x + y, s) = \begin{pmatrix} x & y \\ 32 & 27 \end{pmatrix}$ und damit

$f(y = x - y, \begin{pmatrix} x & y \\ 32 & 27 \end{pmatrix}) = \begin{pmatrix} x & y \\ 32 & 5 \end{pmatrix}$ und $f(x = x - y, \begin{pmatrix} x & y \\ 32 & 5 \end{pmatrix}) = \begin{pmatrix} x & y \\ 27 & 5 \end{pmatrix}$

Mit diesen Ausführungen ist gezeigt, dass ein imperatives, sequentielles Programm als eine *Funktion* aufgefasst werden kann, die Eingabezustände auf Ausgabezustände abbildet. Das Programm muss lediglich aus den oben genannten Bausteinen, insbesondere den Bausteinen Variablenzuweisung, Sequenz, Alternative und Schleife aufgebaut sein.

Wenn nun ein Endzustand mittels Aufgabenstellung oder geeigneter Analyse in einem mathematischen Objekt beschrieben werden kann, dann ist imperatives Programmieren die

Anforderung, eine Funktion zu finden, die mögliche Eingabezustände auf diesen Endzustand abbildet. Weiterhin setzt sich diese Funktion aus elementaren Funktionen wie Alternative, Wertzuweisung, Schleife oder Sequenz zusammen, oder anders: Die gesuchte Funktion kann *algorithmisch* beschrieben werden, womit die Mittel zur Konstruktion einer Lösung gefunden sind.

Der Begriff der Systemkonfiguration soll sich nicht auf imperative Programme beschränken. Trotzdem wird konstatiert:

> **Definition 2:** *Eine Systemkonfiguration ist eine Funktion, die Eingaben auf Ausgaben abbildet. Systemkonfigurationen können mit Mitteln der Algorithmik beschrieben werden.*

2.5.2 Systemkonfiguration und Algorithmik

Systemkonfigurationen werden algorithmisch beschrieben. Was also ist der Unterschied zur Algorithmik? „Ein Algorithmus ist eine präzise, d. h. in einer festgelegten Sprache abgefasste, endliche Beschreibung eines schrittweisen Problemlösungsverfahren zur Ermittlung gesuchter Größen aus gegebenen Größen, in dem jeder Schritt aus einer Anzahl ausführbarer eindeutiger Aktionen und einer Angabe über den nächsten Schritt besteht“ (Pomberger, Rechenberg 2002). Die Definition der Systemkonfiguration scheint sich zunächst einmal nicht davon zu unterscheiden.

2.5.2.1 Algorithmik als fundamentale Idee der Informatik

Nach einer Idee von J.S. Bruner (Bruner 1976) soll sich der Unterricht in einem Schulfach in erster Linie an den „fundamentalen Ideen“ der zugrunde liegenden Wissenschaft orientieren. In ihrer Didaktik der Informatik skizzieren Schubert und Schwill (Schubert, Schwill 2004) die fundamentalen Ideen der Informatik: „*Algorithmik*“, „*Strukturierte Zerlegung*“ und „*Sprache*“.

Eine fundamentale Idee wird hierbei folgendermaßen definiert:
„Eine fundamentale Idee bzgl. eines Gegenstandsbereiches (Wissenschaft, Teilgebiet) ist ein Denk-, Handlungs-, Beschreibungs- oder Erklärungsschema, das

1. in verschiedenen Gebieten des Bereichs vielfältig anwendbar oder erkennbar ist *(Horizontalkriterium)*
2. auf jedem intellektuellen Niveau aufgezeigt und vermittelt werden kann *(Vertikalkriterium)*
3. zur Annäherung an eine gewisse idealisierte Zielvorstellung dient, die jedoch faktisch

möglicherweise unerreichbar ist *(Zielkriterium)*

4. in der historischen Entwicklung des Bereichs deutlich wahrnehmbar ist und längerfristig relevant bleibt *(Zeitkriterium)*
5. einen Bezug zu Sprache und Denken des Alltags und der Lebenswelt besitzt *(Sinnkriterium)*“ (Schubert, Schwill 2004)

Weiter oben wurde definiert: *„Programmieren im Sinne der Informatik heißt, ein Lösungsverfahren für eine Aufgabe so zu formulieren, dass es von einem Computer ausgeführt werden kann“* (Pomberger, Rechenberg 2002). Mit dieser Definition wurden die Punkte oder Tätigkeiten Aufgabenanalyse, Lösung und Formulierung identifiziert, die man im Großen und Ganzen mit den drei fundamentalen Ideen der Informatik nach Schwill (Schwill 1993) in Beziehung setzen kann. Die Aufgabenanalyse in der Konkretisierung einer objektorientierten Modellierung fällt in den Bereich der fundamentalen Idee *“strukturierte Zerlegung“*. Die Formulierung der Lösung, so dass ein Computer sie ausführen kann, entspricht der fundamentalen Idee *„Sprache“* und kann mit Codierung gleichgesetzt werden. Das Lösungsverfahren letztlich entspricht der Idee der *„Algorithmisierung“*, so dass Programmieren im weitesten Sinne alle wichtigen Bereiche und Ideen der Informatik umfasst.

Schubert und Schwill vereinigen unter der fundamentalen Idee *„Algorithmisierung“* folgende Ideen gemäß Abbildung 4:

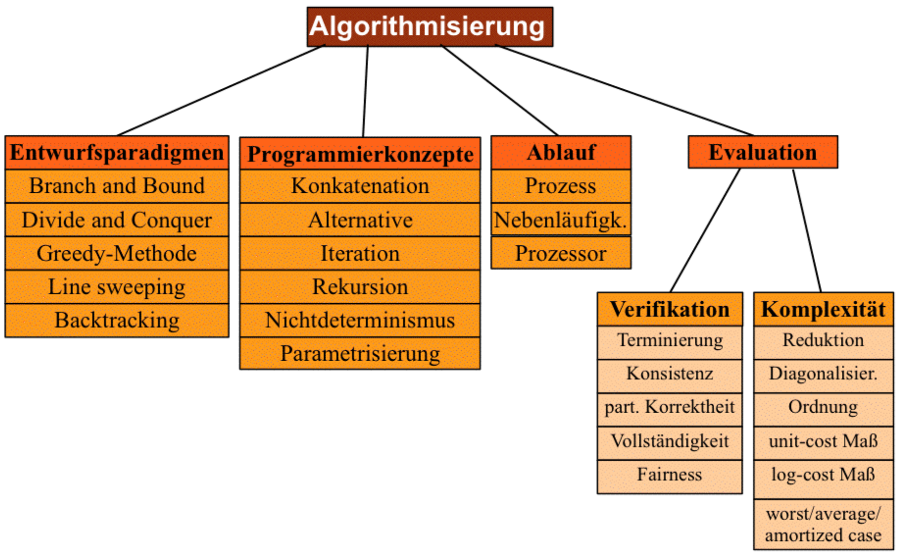


Abbildung 4: Die fundamentale Idee „Algorithmisierung“, Bild unter der URL: http://www.informatica-didactica.de/InformaticaDidactica/Modrow2006, Zugriff: 07.07.2009

Im folgenden Abschnitt soll nun gezeigt werden, wie sich der in dieser Arbeit eingeführte Begriff Systemkonfiguration gegen „Algorithmisierung“ abgrenzt und welche Gemeinsamkeiten es gibt.

2.5.2.2 Systemkonfiguration in Abgrenzung zur Algorithmik

Nach Abelson, Sussman und Sussman (Abelson, Sussman, Sussman 1996) besitzt jede leistungsfähige Programmiersprache drei Dinge:

- *elementare Ausdrücke*, die die einfachsten Einheiten der Sprache repräsentieren,
- *Mittel zur Kombination*, mit denen zusammengesetzte Elemente aus einfachen konstruiert werden, und
- *Mittel zur Abstraktion*, mit denen zusammengesetzte Elemente benannt und als Einheiten behandelt werden können.

Im Vergleich mit Kapitel 2.5.1 sieht man, dass bei imperativen Programmen z. B. die Zuweisung einen elementaren Ausdruck darstellt. Die Sequenz oder Alternative sind Mittel zur Kombination dieser elementaren Ausdrücke.

Algorithmik als fundamentale Idee muss in verschiedenen Bereichen der Informatik auffindbar sein *(Horizontalkriterium)*. Diese Eigenschaft „erbt“ auch der Begriff der Systemkonfiguration. Zur Veranschaulichung der folgenden Überlegungen wird ein System aus Schaltern und Lampen untersucht. Die Schalter werden zur digitalen Eingabe und die Lampen als Visualisierung der Ausgabe digitaler Werte verwendet.

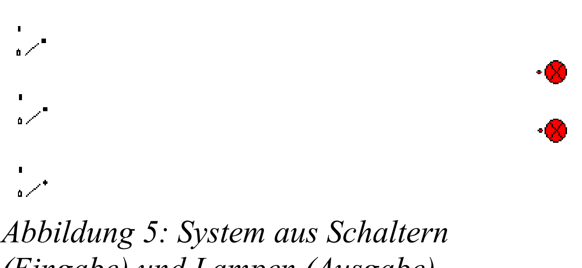


Abbildung 5: System aus Schaltern (Eingabe) und Lampen (Ausgabe)

An dieser Stelle wird im Hinblick auf eine spätere Definition festgehalten, dass das System auch ohne Konfiguration lauffähig ist. Die Lampen und Schalter stehen zwar in keiner Beziehung zueinander, das System könnte dennoch verwendet werden, es produziert keine „Fehler“. Soll dieses System nun konfiguriert werden, finden Gatter (UND, ODER, NICHT) und Drähte Verwendung. Nach Abelson et al. (Abelson, Sussman, Sussman 1996) bilden die Gatter (UND, ODER und NICHT) die elementaren Operationen und die Drähte Mittel zur Kombination derselben.

Bei Aufgaben der Entwicklung von Schaltnetzen im Unterricht werden auch Mittel zur Abstraktion eingesetzt. Die Funktionsweise eines Halbaddierers beispielsweise wird einmal mit den Grundgattern UND, ODER und NICHT geschaltet, benannt und im weiteren Verlauf des Unterrichts als Halbaddiererbaustein verwendet. Damit sind die Anforderungen an eine Programmiersprache gegeben. Eine mögliche Systemkonfiguration in dieser Sprache ist folgende:

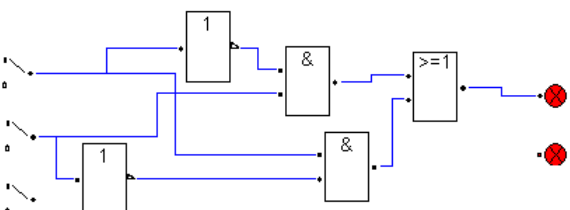


Abbildung 6: Systemkonfiguration in einem System von Schaltern und Lampen

Aus dieser Konfiguration wird nun folgendes ersichtlich:

- Jede elementare Operation (jedes Gatter) kann an *beliebiger* Stelle eingefügt werden, das System bleibt lauffähig und ist immer in irgendeiner Weise konfiguriert, wenn auch nicht sinnvoll. Es gibt immer „ein Ergebnis“.
- Die elementaren Operationen können *beliebig* kombiniert werden, das System bleibt lauffähig.
- Jedes beliebige Schaltnetz kann mit den Grundgattern UND, ODER und NICHT realisiert werden. Gibt eine Aufgabenstellung an, bei welcher Eingangsbelegung welche Ausgabe erfolgen soll, dann kann mit der Konstruktion einer „Disjunktiven Normalform“ oder „Konjunktiven Normalform“ ein solches Schaltnetz systematisch erarbeitet werden.
- Die elementaren Operationen sind Funktionen, die Eingabewerte auf Ausgabewerte abbilden.

Beispiel 9:

Sei *B={0,1}*. Dann definieren wir z. B. die elementare Funktion *UND* wie folgt:

$$f : B \times B \to B \text{ mit } f(b_1, b_2) = \begin{cases} 1 & \text{falls } b_1 = b_2 = 1 \\ 0 & \text{sonst} \end{cases}$$

Diese Beobachtungen fließen in eine weitere Konkretisierung des Begriffs Systemkonfiguration ein.

Definition 3: *Systeme, die konfiguriert werden sollen, verfügen über Eingaben und Ausgaben. Zur Konfiguration dieser Systeme werden sogenannte Basiselemente verwendet, die Eingabewerte auf Ausgabewerte abbilden. Weiterhin gibt es Operationen auf diesen Basiselementen, die beliebig kombiniert werden können. Jede beliebige endliche Kombination von Anwendungen der Operationen und Basiselementen muss wieder eine gültige Konfiguration des Systems ergeben, das System bleibt weiterhin lauffähig.*

Damit enthalten Systemkonfigurationen einen Satz beliebig kombinierbarer Funktionen, die Eingabewerte auf Ausgabewerte abbilden.

Hervorzuheben ist in obiger Definition, dass die Basiselemente den Fluss von Eingaben in Ausgaben beibehalten. Dass Schaltnetze oder boolesche Ausdrücke mit den Grundoperationen UND, ODER und NICHT diese Anforderung erfüllen, hat das obige Beispiel gezeigt.
Da $A \vee B = \overline{\overline{A \wedge A} \wedge \overline{B \wedge B}}$, $A \wedge B = \overline{\overline{A \wedge B} \wedge \overline{A \wedge B}}$ und $\overline{A} = \overline{A \wedge A}$, ist auch die NAND-Verknüpfung ein mögliches einziges Basiselement.

Auch Schubert und Schwill skizzieren diese Idee der Erzeugendensysteme und sprechen von einer Orthogonalisierung eines Objektbereichs (Schubert, Schwill 2004).

Die Programmiersprache Java, die ebenfalls als Codierung eines Algorithmus verwendet werden kann, entspricht *nicht* den Anforderungen aus Definition 3. Operationen lassen sich hier nicht beliebig kombinieren, ohne dass Fehler erzeugt werden. Beispielsweise kann man im Rumpf einer `while`-Schleife keine Klassen deklarieren. Java kann damit nicht als Beschreibungsmittel einer Systemkonfiguration gewählt werden.

Imperative Sprachen, wie sie in Kapitel 2.5.1 angegeben wurden, erfüllen aber die Anforderungen an Möglichkeiten, Systemkonfigurationen darzustellen. „Die Strukturen Zuweisung, Konkatenation und `while`-Schleife bilden eine Basis der Kontrollstrukturen. Jede andere Anweisung lässt sich durch diese drei Typen darstellen“ (Schubert, Schwill 2004). Auch *funktionale Programmiersprachen*, wie sie in Kapitel 3 untersucht werden, erfüllen diese Anforderungen. „Die Grundoperationen bei Sprachen, die sich am λ-Kalkül orientieren, sind Abstraktion (= Definition einer Funktion mit Parametern), Applikation (= Aufruf einer Funktion mit aktuellen Parametern) und Substitution (= Parameterersetzung)“ (Schubert, Schwill 2004). Die *booleschen Funktionen* UND,

ODER, NICHT können als Mittel zur Darstellung einer Systemkonfiguration angesehen werden, wie im Beispiel gezeigt. Ebenso bildet die universelle *Turingmaschine* die (einelementige) Basis der Klasse aller Turingmaschinen (Schubert, Schwill 2004). An späterer Stelle in dieser Arbeit werden die angeführten Beschreibungsmittel von Interesse.

Die Mächtigkeit der Algorithmen ist durch Definition 3 nicht beschränkt. Auch wenn es den Anschein hat, als sei Definition 3 sehr einschränkend, können mit Systemkonfigurationen alle Algorithmenbausteine beschrieben werden. So lassen sich neben der universellen Turingmaschine auch die primitiv-rekursiven und μ-rekursiven Funktionen orthogonalisieren und bilden ein Erzeugendensystem. „Es gibt eine Reihe von Grundfunktionen (z. B. konstante Funktion 1, Nachfolgerfunktion) und eine Reihe von Operationen (z. B. Komposition, Einsetzung von Funktionen, μ-Operator), mit der man jede primitiv-rekursive bzw. μ-rekursive Funktion erzeugen kann" (Schubert, Schwill 2004). Dass jeder Algorithmus damit nach wie vor angegeben werden kann, zeigt die Churchsche These:
„Churchsche These: Die durch die formale Definition der Turing-Berechenbarkeit (äquivalent: μ-Rekursivität) erfasste Klasse von Funktionen stimmt genau mit der Klasse der im intuitiven Sinne berechenbaren Funktionen überein" (Schöning 1992).

Definition 3 schränkt also nicht die Mächtigkeit der Algorithmen, wohl aber die *Beschreibungsmittel* zur Angabe des Algorithmus ein, weshalb die Einführung eines neuen Begriffs Systemkonfiguration nach Meinung der Autorin gerechtfertigt ist.

Die Einschränkung der Beschreibungsmittel hat keinen Einfluss darauf, dass auch der Begriff der Systemkonfiguration dem Horizontal- und Vertikalkriterium nach Schubert und Schwill genügt, wie folgende drei Beispiele zeigen. Didaktisch gesehen hat diese Einschränkung aber wichtige Auswirkungen, wie diese Arbeit im Verlauf zeigt.

Beispiel 10:

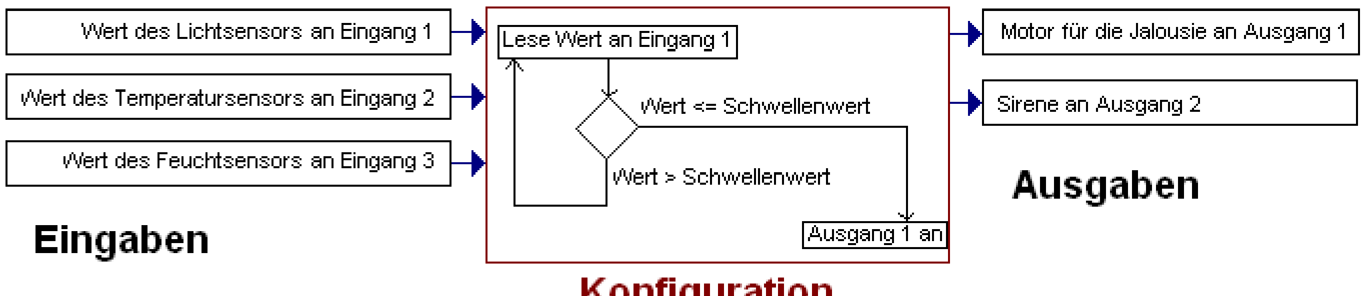


Abbildung 7: Konfiguration eines Systems zur Jalousiesteuerung

In diesem Beispiel besteht das System aus drei Sensoren und zwei Aktoren. Die Konfiguration bewirkt, dass bei einer bestimmten Lichtintensität ein Motor angesteuert wird, der eine Jalousie bewegt. Basiselemente sind die Funktionen: *LiesWertAnEingang1* (die Ausgabe wird hierbei nicht verändert) oder die Funktion: *Ausgang1An* (bei jeder möglichen Eingabe). Ein Mittel zur Kombination ist hier z. B. die Schleife. Neben der in Abbildung 7 dargestellten Konfiguration könnte das System auch so spezialisiert werden, dass eine Jalousie bei Dunkelheit, bei Temperaturen unter Null oder bei Regen herunter gelassen wird, um die Wärmeabgabe nach außen zu minimieren. Bei Temperaturen unter dem Gefrierpunkt und Niederschlag meldet eine Sirene außerdem, dass ein Einfrieren des Motors zu befürchten ist. Auch eine Konfiguration, die nur die Operation *LiesWertAnEingang1* beinhaltet, ist eine gültige Konfiguration.

Auch in der „Theoretischen Informatik", die in der Schule hauptsächlich formale Sprachen, Grammatiken, endliche Automaten, Kellerautomaten und Turingmaschinen umfasst (siehe dazu EPA 2004), kann das Konzept erkannt werden, die Konfiguration eines Systems z. B. mit Hilfe der Angabe einer Turingmaschine darzustellen, die aus elementaren Funktionen zusammengesetzt oder konstruiert ist:

Beispiel 11:

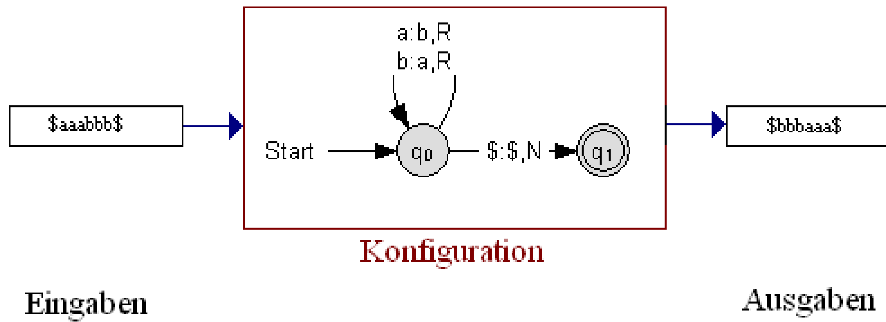


Abbildung 8: Konfiguration eines Systems mittels Turingmaschine

Die elementaren Funktionen einer Turingmaschine wie im Beispiel sind das Schreiben eines Zeichens und eine Kopfbewegung. Durch Zustände und die Angabe zu lesender Zeichen können Bedingungen angegeben werden, unter denen die elementaren Funktionen ausgeführt werden. Wiederholungen und Sequenzen der elementaren Funktionen sind ebenfalls darstellbar, in der oben verwendeten Syntax einer Turingmaschine z. B. durch Pfeile in die verschiedenen Zustände. Je nachdem, wie diese Funktionen nun zusammengesetzt werden, werden unterschiedliche Ausgaben produziert, wird das System unterschiedlich konfiguriert.

Das Konzept der Systemkonfiguration ist somit in den verschiedenen Bereichen der Informatik erkennbar *(Horizontalkriterium)*. Das Prinzip lässt sich außerdem auch auf unterschiedlichen Niveaus finden *(Vertikalkriterium)*.

Beispiel 12:
Eine Schülerin oder ein Schüler hat eine fertige PowerPoint-Präsentation vorliegen. Sie beschreibt das System und ist funktionsfähig. Jetzt kann der Schüler oder die Schülerin durch die Auswahl von Animationen bestimmte Eingaben (Mausklicks) in diesem System mit bestimmten Ausgaben (Textfeldern) verknüpfen, so dass die Inhalte des Textfeldes z. B. von rechts oben hereingerieselt kommen. Eine andere Konfiguration wäre das Eindrehen des Textes aus der linken unteren Ecke. Der Schüler oder die Schülerin konfiguriert das System PowerPoint-Präsentation gemäß seinen oder ihren Wünschen. Dabei führt jede Konfiguration zu einer lauffähigen, wenn auch nicht immer sinnvollen Präsentation.

Eine Ampelsteuerung ist als Schaltung, die in einem System mit Lampen, Schaltern, Gattern und Drähten zu realisieren ist, dabei ungleich komplexer.

2.6 Systemkonfiguration als didaktischer Rahmen im Anfangsunterricht

Die Einschränkungen der Beschreibungsmittel von Systemkonfigurationen auf ein System aus beliebig kombinierbaren Funktionen gemäß Definition 3 führt zu wesentlichen Vorteilen im Unterricht mit Schülerinnen und Schülern der Sekundarstufe I.

- Die Lernenden entwerfen eine Konfiguration als Kombination von Operationen auf Basiselementen. Eine Konfiguration wird so nach dem Baukastenprinzip zusammengesetzt.
- Da nur so wenige Basiselemente und Operationen wie nötig zur Verfügung stehen (siehe Definition 3) und diese beliebig kombiniert werden können, können die schwächeren

Schülerinnen und Schüler, die keine eigene Lösungsstrategie entwickeln, durch *Experimentieren* zum gewünschten Ergebnis kommen.

- Das System bleibt immer lauffähig. Egal, welche Operation die Schülerinnen und Schüler anwenden, sie haben immer ein lauffähiges Produkt. Auch wenn das nicht ganz ihren Wünschen entspricht, „funktioniert" es doch auf irgendeine Weise.

Die sich daraus ergebenden Vorteile sind in der „Stärkung des Schüler-Ichs" (vgl. Heymann 1996) zu sehen. Schülerinnen und Schüler sind in der Lage, auch wenn sie keine Lösungsstrategie und keinen Algorithmus entwickeln können, doch ein lauffähiges Produkt zu erzeugen und durch „Versuch und Irrtum" vielleicht sogar zu einer richtigen Lösung zu kommen. Dieser Punkt ist die Grundvoraussetzung dafür, dass eine eigenständige Algorithmensuche „für Alle" überhaupt möglich ist. Erfahrungen im Unterricht zeigen, dass bei einem Programmierunterricht „für Alle" die Suche nach Lösungsalgorithmen sehr polarisierend wirkt. Während einige Schülerinnen und Schüler solche Aufgaben sofort lösen, „scheitern" andere Lernende daran auf ganzer Linie. Einige Unterrichtskonzepte schränken deshalb die eigenständige Algorithmensuche ein, siehe u. a. den Ansatz „Dekonstruktion von Systemen" (nach Hampel, Magenheim, Schulte 1999). Mit der didaktischen Reduktion der Systemkonfiguration ist aufgrund der wenigen Basiselemente und der beliebigen Kombinierbarkeit der Operationen eine eigenständige Algorithmensuche für alle Schülerinnen und Schüler möglich. Kapitel 4 widmet sich diesem Aspekt detaillierter.

Die vorliegende Arbeit zeigt in Kapitel 3, dass es unter dem Anspruch der Allgemeinbildung sinnvoll ist, sich auf technische Systeme zu konzentrieren. Unterrichtsbeispiele in Kapitel 6 zeigen, dass Schülerinnen und Schülern, die mit der Systemkonfiguration technischer Systeme vertraut sind, ein Transfer auf unbekannte Techniksysteme ihrer Lebenswelt in der Form gelingt, dass sie befähigt werden, die Funktionalität technischer Systeme ihrer Umwelt, die nicht Gegenstand des Unterrichts waren, *algorithmisch* zu beschreiben. Den Schülerinnen und Schülern gelingt trotz der gemachten Einschränkungen der Beschreibungsmittel das Durchdringen, die Verinnerlichung und das Verständnis für die zugrunde liegenden Algorithmenbausteine.

An dieser Stelle ist hervorzuheben, dass Techniksysteme in der Realität die Basiselemente und zu verwendenden Operationen einer Systemkonfiguration sogar noch weiter einschränken (siehe dazu Kapitel 3).

2.7 Systemkonfiguration und Modellierung

In Kapitel 2.4 wurden Unterrichtskonzepte aufgezeigt, die ihren Schwerpunkt auf eine Modellierung oder einen Systementwurf legen. In den Arbeiten von Hubwieser wurde gezeigt, wie bereits im Anfangsunterricht der Sekundarstufe I objektorientierte Modellierung unterrichtet werden kann (vgl. dazu Hubwieser 2000). In dieser Arbeit stehen nicht der Systementwurf und die Modellierung im Blickpunkt des Interesses, sondern die Konstruktion einer Lösung, eines Algorithmus bei gegebenem Endzustand als Konfiguration eines bestehenden Systems. Dies ist ein anderer Ansatz.

Dabei sollten nach Ansicht der Autorin nicht beide Schwerpunkte im Anfangsunterricht der Sekundarstufe I gleichzeitig gelehrt werden, weil sie gegenläufige Kompetenzen von den Schülerinnen und Schülern fordern. Das wird wie folgt begründet:

Beginnen Lernende mit der Analyse, dann zunächst relativ unspezifisch. In mehreren Schritten werden die ersten vagen Modelle immer weiter verfeinert, bis sich die Lösung des Problems ergibt. Es handelt es sich also „um eine sukzessive Konkretisierung von abstrakt formulierten Lösungsideen“ (Pomberger, Rechenberg 2002). Die Analyse eines Problems wird immer konkreter, bis Klassen und Objekte identifiziert werden können, und bildet somit ein klassisches Top-down-Verfahren. Die Problemlösung besteht in der Modellierung eines Systems, in dem Objekte, die über Nachrichtenströme gegenseitige Methodenaufrufe provozieren, miteinander in Beziehung stehen. Für diesen Anforderungsbereich werden von den Schülerinnen und Schülern also *Top-down-Strukturierungsfähigkeiten* gefordert. Die Entwurfstechnik entspricht einem „zielgerichteten Entwurf“ (Pomberger, Rechenberg 2002).

Bei der Konstruktion der Lösung gilt es, die Methoden mit Inhalt zu füllen, damit das System die gewünschte Funktionsweise liefert. Bei einer Systemkonfiguration werden bekannte elementare Operationen mit eindeutiger Funktionalität wie z. B. Zuweisung oder Berechnungen mit Mitteln der Kombination, z. B. Sequenz oder Alternative, so lange zu komplexeren und mächtigeren Operationen nach dem Baukastenprinzip zusammengesetzt, bis die Lösung der Aufgabe gefunden ist. Für diesen Anforderungsbereich werden von den Schülerinnen und Schülern *Bottom-up-Konstruktionsfähigkeiten* gefordert. Die Entwurfstechnik entspricht hier einem „kompositionellen Entwurf“ (Pomberger, Rechenberg 2002). Werden Systementwurf und Systemkonfiguration gleichzeitig gelehrt, dann muss man sich darüber im Klaren sein, dass von Programmieranfängern der Sekundarstufe I gleichzeitig die Aneignung zweier für sie neuer, gegenläufiger Kompetenzen

gefordert wird: *Top-down-Strukturierungsfähigkeiten* und *Bottom-up-Konstruktionsfähigkeiten*[6]. Obgleich eine enge Verzahnung unumgänglich ist, sollten die ersten Ansätze nach Meinung der Autorin erst *nacheinander* gelehrt werden.
An dieser Stelle sollen Gründe aufgezeigt werden, die zeigen sollen, dass es sinnvoll sein kann, zunächst bestehende Systeme zu konfigurieren, *bevor* neue unbekannte Systeme entworfen bzw. modelliert werden.

Begrenzt man in einem ersten Schritt die Anforderungen der Programmierung didaktisch auf die Konfiguration bestehender Systeme, die intuitiv zu erfassen oder allgemein bekannt sind, dann sprechen folgende Gründe für dieses Vorgehen:

1. *Modellbildung setzt Kenntnisse in der Programmierung voraus*. Wenn z. B. das System eines Betriebes modelliert werden soll, dann braucht man einen Zweck und ein Ziel des Modells. Ein Informatiker, der Betriebsprozesse später rechnergestützt verbessern will, entwirft ein anderes Modell des Betriebs als ein Psychologe, der die Betriebsführung über die Verbesserung der Mitarbeiterzufriedenheit beraten will. Um Verbesserungsvorschläge der Mitarbeiterzufriedenheit zu geben, muss man erst einmal definiert haben, was Mitarbeiterzufriedenheit ist. Um im Rahmen der Programmierung geeignete Modelle entwickeln zu können, sollte man Erfahrungen in der „Programmierung im Kleinen" haben. Um *neue* Systeme entwerfen zu können, sollte man Erfahrungen in der Konfiguration *vorhandener* Systeme haben.
2. Systemkonfigurationen bewegen sich in einem fest vorgegebenen Rahmen (System), was die Komplexität reduziert. Sie lassen sich nach dem Baukastenprinzip zusammensetzen. Da es nur wenige Basiselemente gibt, kann mit diesen experimentiert werden, was eine weitere Lösungsstrategie beinhaltet. Das Baukastenprinzip unterstützt die Schülerinnen und Schüler bei der Konstruktion und visualisiert außerdem den konstruierten Algorithmus in geeigneter Weise. Ein Systementwurf oder eine Modellierung beschäftigt sich hingegen mit unendlich vielen Möglichkeiten der Kombination und ist dadurch wesentlich weniger überschaubar.
3. Eine Systemkonfiguration bedeutet, ein vorhandenes System zu spezialisieren. Man erzeugt dabei ein *konkretes Produkt.* Auch dies ist für Schülerinnen und Schüler der Sekundarstufe I eine wichtige Voraussetzung im Unterricht, wie in nachfolgenden Kapiteln gezeigt werden wird.

[6] Im Meyer-Lexikon (Meyer 1990) findet man zur Frage nach „Programmierung" folgende Definition: „Bei der strukturierten Programmierung werden, ausgehend von einem vorgegebenen Problem, geeignete Abstraktionen herausgearbeitet und in einer Hierarchie angeordnet. Die strukturierte Programmierung umfasst die Verfeinerung vom (abstrakten) Modell zum (konkreten) Problem (Top-down-Programmierung) sowie das Zusammensetzen vorhandener Programmbausteine zu Programmen (Bottom-up-Programmierung)", was diese Aussage stützt.

Systemkonfigurationen erzeugen in geeigneten Lernumgebungen von Beginn an lauffähige Produkte, bei denen sich etwas tut. Schalten wir Sensoren und Aktoren eines Systems entsprechend einer Aufgabenstellung geeignet zusammen, dann kann der Lernende etwas vorführen und zeigen, dass sich z. B. etwas bewegt, was Produktstolz unabhängig vom Urteil des Lehrers erzeugt. Systemkonfigurationen sind damit zweckorientiert und in diesem Sinne technisch. Beginnt der Unterricht in Klasse 6 hingegen mit der Beschreibung von Objekten und Klassen, um einen „Grundstein für den Aufbau angemessener mentaler Modelle und die Verwendung einer sauberen, ausdrucksstarken Terminologie in den späteren Jahren“ (Hubwieser 2000) zu legen, gestaltet sich die Vermittlung des Sinns dieser Aufgaben für Schülerinnen und Schüler, die sich in der Oberstufe nicht mehr mit Informatik beschäftigen, nach Meinung der Autorin schwieriger.

4. Die Fragen des Alltags, mit denen sich Schülerinnen und Schüler beschäftigen, sind meist sogenannte „Wie“-Fragen. Wie funktioniert ein Toaster? Wie funktioniert die Waschmaschine? Sie erwarten Antworten auf die Funktionalität der Geräte. Die Funktionalität oder Konfiguration wird aber algorithmisch beschrieben. Schülerinnen und Schüler fragen nicht: Was ist ein Toaster? Was ist eine Waschmaschine? Das System Toaster oder Waschmaschine ist bekannt und intuitiv zu erfassen.

An dieser Stelle soll angemerkt werden, dass es auch kritische Stimmen bzgl. des objektorientierten Paradigmas gibt. Börstler (Börstler 2007) z. B. fragt: „Objektorientiertes Programmieren – Machen wir irgendetwas falsch?“ Er beschreibt die immer schlechter werdenden Ergebnisse in der Vorlesung „Einführung in die Programmierung“ an Hochschulen, denen objektorientierte Programmiersprachen zugrunde liegen. In diesem Zusammenhang gehen auch Rabel und Oldenburg (Rabel, Oldenburg 2009) auf eine Befragung von Schülern und Studenten der Informatik ein und schlussfolgern, dass „die Vermittlung der Objektorientierung noch nicht optimal ist. Erst Studierende bewerten – wohl vor dem Hintergrund der Erfahrung auch mit größeren Projekten – diese Konzepte höher“ (Rabel, Oldenburg 2009).

2.8 Zusammenfassung

Diese Arbeit begründet einen neuen Ansatz für den Anfangsunterricht in der Programmierung „für Alle“. Da im Aufgabenfeld der Programmierung sehr verschiedene Kompetenzen von den Schülerinnen und Schülern gefordert werden, die alle Grundideen der Informatik vereinen, scheint es angemessen, den Anforderungsbereich zunächst didaktisch zu reduzieren. Diese Arbeit hält das

didaktisch-methodische Handlungsschema „Systemkonfiguration“ als didaktische Reduktion im Anfangsunterricht „Programmieren für Alle“ für sinnvoll. Als Endergebnis wird Systemkonfiguration folgendermaßen definiert:

Definition Systemkonfiguration:
Ist ein an sich bereits lauffähiges System mit Eingaben und Ausgaben gegeben, welches intuitiv zu erfassen oder ausreichend analysiert ist, dann bedeutet Systemkonfiguration, die Funktionalität dieses Systems so zu verändern, dass das System für eine bestimmte Aufgabe spezialisiert wird. Eine Systemkonfiguration ist dabei eine Funktion, die Eingaben auf Ausgaben abbildet. Als Beschreibungsmittel einer Konfiguration werden sogenannte Basiselemente verwendet, die ihrerseits Eingabewerte auf Ausgabewerte abbilden. Weiterhin gibt es Operationen auf diesen Basiselementen, die beliebig kombiniert werden können. Jede beliebige endliche Kombination von Anwendungen der Operationen und Basiselementen muss wieder eine gültige Konfiguration des Systems ergeben.

3 Inhaltliche Ausgestaltung des Programmierens in der Sekundarstufe I

Im Gymnasium werden nicht nur Informatikerinnen und Informatiker, sondern zukünftige Bankkaufleute, Tierärzte, Förster und Rechtsanwälte ausgebildet. Ein Unterricht für alle muss somit auch im Informatikunterricht, und hier speziell in einer Unterrichtseinheit Programmieren, dem Anspruch der Allgemeinbildung genügen. Nach der Festlegung auf den didaktischen Rahmen einer Systemkonfiguration stehen in diesem Kapitel die Art der Systeme und damit die Art der Aufgaben im Mittelpunkt der Betrachtungen, die unter dem Aspekt der Allgemeinbildung inhaltliche Relevanz haben.

3.1 Programmierunterricht und Weltorientierung

Hans Werner Heymann (Heymann 1996) hat verschiedene Aspekte erarbeitet, von denen Anforderungen an allgemeinbildende Unterrichtsinhalte abgeleitet werden können. In seinem Allgemeinbildungskonzept nennt er die Punkte „Lebensvorbereitung“, „Stiftung kultureller Kohärenz“, „Weltorientierung“, „Anleitung zum kritischen Vernunftgebrauch“, „Entfaltung von Verantwortungsbereitschaft“, „Einübung in Verständigung und Kooperation“, sowie „Stärkung des Schüler-Ichs“.

In diesem Kapitel sollen besonders der Bereich der „Weltorientierung“ aus Heymanns Anforderungskatalog herausgegriffen und Forderungen abgeleitet werden, die sich auf die Inhalte einer Unterrichtseinheit Programmierung auswirken. Heymann beschreibt diesen Punkt folgendermaßen: „Die Schüler sollen einen Überblick haben, die Erscheinungen um sich herum einzuordnen wissen, sie zueinander in Beziehung setzen können, über ihren engeren Erfahrungshorizont hinaus über die Welt „Bescheid wissen““ (Heymann 1996). Helmut Witten schreibt dazu: „Es ist daher Aufgabe einer Weltorientierung durch informatische Bildung, die Informationstechnik in den alltäglichen Anwendungen sichtbar und verstehbar zu machen“ (Witten 2003).

Was sind für Schülerinnen und Schüler nun alltägliche Anwendungen der Informationstechnik? Befragen wir die Lernenden, dann sind das sicherlich der MP3-Player und der DVD-Player, das Handy und der Getränkeautomat, der Barcodescanner im Supermarkt genauso wie Waschmaschine, Wäschetrockner, Klimaanlage und die Sitzheizung im Auto. Informatikunterricht soll also die *technischen Systeme*[7] aus der Lebenswelt der Schülerinnen und Schüler oder kurz die

[7] In Meyer (Meyer 2007) wird definiert: „Anstelle der schlecht abgrenzbaren Ausdrücke Maschine, Gerät, Apparat ist heute als allgemeiner Begriff das *technische System* gewählt worden. Es ist durch die Funktion gekennzeichnet, Stoff (Masse), Energie und/oder Information umzuwandeln, zu transportieren und/oder zu speichern.“

„Alltagsgeräte“ begreifbar machen, wenn der Anspruch der Allgemeinbildung in einem Unterricht „für Alle“ gewahrt sein soll.

Technische Systeme ermöglichen eine spezialisierte Anwendung, die sich aus ihrer Ausstattung, also den Einzelteilen ihrer Herstellung, und der internen Programmierung der Mikroprozessoren ergibt. Das System, welches einer Systemkonfiguration zugrunde liegen muss, ist gegeben durch die Art der Eingaben und Ausgaben. In *technischen Systemen* kann man diese näher spezifizieren und somit eingrenzen.

3.2 Beschreibung eines Techniksystems

Techniksysteme aus der Lebenswelt der Schülerinnen und Schüler enthalten wesentlich *Sensoren*, die physikalische Größen messen. Es gibt Feuchtesensoren und Temperatursensoren im Wäschetrockner, Lichtsensoren im Strichcodescanner oder Druck- und Geräuschsensoren im Blutdruckmessgerät. Alle diese Sensoren liefern fortlaufend Eingabewerte, die verarbeitet werden müssen. Die Ergebnisse der Verarbeitungsschritte sind Ausgabedaten für die sogenannten *Aktoren*. Im Wäschetrockner kann ein Motor angesteuert werden, der die Trommel dreht, außerdem eine Heizspule oder ein Gebläse. Roboterarme können angesteuert werden und Drehbewegungen ausführen, ein System kann piepen, Texte in einem Display anzeigen oder sich ausschalten. Aktoren sind in technischen Systemen damit häufig Motoren, Lampen (Anzeigen) oder Summer.

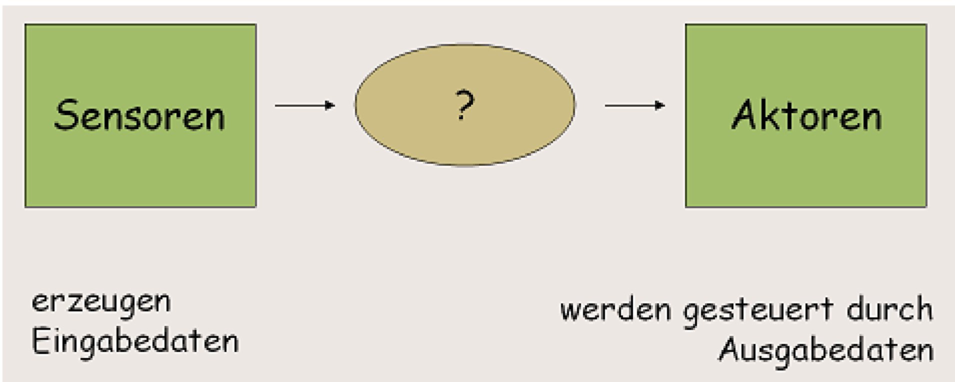


Abbildung 9: Schema eines Techniksystems

In den Techniksystemen der Lebenswelt der Schülerinnen und Schüler sind die Sensoren und Aktoren schnell identifiziert. Beschäftigt man sich z. B. mit einem automatischen Blutdruckmessgerät, dann identifiziert man als Sensoren einen Geräuschsensor, der Herztöne registriert, und einen Drucksensor, der den aktuellen Druck der Armmanschette misst. Weiterhin gibt es als Aktoren in diesem System ein „Gebläse“, welches die Manschette aufblasen kann, sowie eine Textanzeige.

Meist ist die Zahl der Sensoren und Aktoren so begrenzt und schnell erfassbar, dass eine Systemmodellierung oder ein Systementwurf überflüssig wird, denn das System ist den Schülerinnen und Schülern aus ihrem täglichen Umgang vertraut. Eine Modellierungsphase kann außer Acht gelassen werden. In den Blickpunkt des Interesses gerät vielmehr die Frage: „*Wie funktioniert* ein Blutdruckmessgerät? *Wie funktioniert* dies technische System?“, was eine Frage nach der Systemkonfiguration darstellt. Das „?“ aus Abbildung 9 muss mit den Beschreibungsmitteln der Systemkonfiguration formuliert werden.[8]

Diese Arbeit beschränkt sich im Folgenden auf die Konfiguration *technischer* Systeme. Diesen Allgemeinbildungsbezug findet man auch bei Reichert et al. (Reichert, Nievergelt, Hartmann 2005), die sich mit Programmierumgebungen für Anfänger auseinandersetzen. Reichert et al. fordern *„alltagsorientierte Aufgabenstellungen“*. Sie schreiben: „[...] Besonders geeignet sind Problemstellungen, die kognitiv mit Bewegungen in der realen Welt in Verbindung gebracht werden können. [...]“ (Reichert, Nievergelt, Hartmann 2005).

In den folgenden Abschnitten werden anhand verschiedener Beispiele bestimmte Aspekte der Konfiguration technischer Systemen näher betrachtet. Eine Formalisierung erfolgt im Anschluss daran.

3.3 Konfigurationen in einem Techniksystem

Zunächst sollen zwei Beispiele Konfigurationen in einem Techniksystem konkretisieren.

Beispiel 12:

Sollen Schülerinnen und Schüler, die in einem Technikfach Informatik unterrichtet werden, die Funktionalität oder Konfiguration des oben genannten Systems Blutdruckmessgerät erläutern, erwartet der Lehrende, dass die Schülerinnen und Schüler die Funktionalität durch einen Algorithmus wie folgt angeben können: Die Manschette wird bis zu einem bestimmten oberen Wert aufgeblasen, der in der Regel nicht überschritten wird. Dann wird die Luft so lange aus der Manschette abgelassen, bis der Geräuschsensor Herztöne registriert. Der Drucksensor gibt jetzt seinen Wert an, der systolischer Druck genannt wird. Anschließend wird weiter Luft aus der Manschette gelassen, bis der Geräuschsensor keine Herztöne mehr registriert. Dieser jetzt

[8] Vielleicht erscheint das Beispiel Blutdruckmessgerät an dieser Stelle ungünstig. In Kapitel 6 wird gezeigt, dass es Schülerinnen und Schülern aber tatsächlich gelingt, ohne vorhergehenden Unterricht in (objektorientierter) Modellierung die Sensoren und Aktoren eines Techniksystems zu identifizieren.

gemessene aktuelle Druck wird als diastolischer Wert angezeigt.[9]

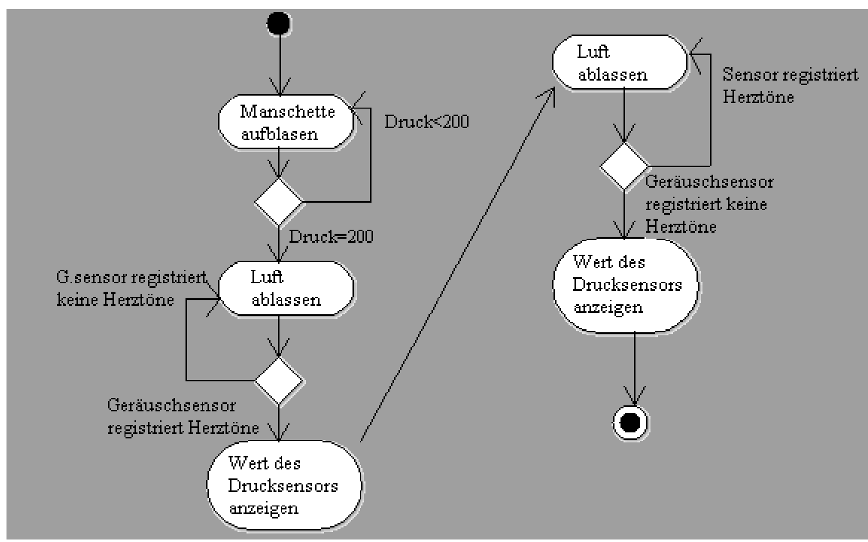


Abbildung 10: Funktionsweise eines Blutdruckmessgerätes

Beispiel 13:

Ein landwirtschaftliches Pflanzenschutzgerät lässt sich z. B. auf 200 l/ha einstellen. Je nachdem, wie schnell die Zugmaschine fährt, wird der Düsendruck entsprechend geregelt. Außerdem bleibt der Abstand der Düsen zum Boden gewahrt, auch in Schräglagen am Hang. Jede Düse hat stets denselben Bodenabstand. Die Schülerinnen und Schüler stehen vor der Aufgabe, die Eingaben des Abstandsmessers (Ultraschallsensor) und des Geschwindigkeitsmessers so zu verknüpfen, dass Pumpen im Pflanzenschutzgerät und Motoren zur Änderung der Neigung des Gerätes als Aktoren im Sinne der Aufgabe angesteuert werden.

An dieser Stelle sei angemerkt, dass hier zwei disjunkt parallele Prozesse betrachtet werden, die Regelung des Bodenabstands der Düsen und des Düsendrucks. Wir gehen später auf diese Problematik wieder ein.

Auch wenn durch das Handlungsschema der Systemkonfiguration und ihrer Definition in Kapitel 2 Unterschiede zu universellen *objektorientierten Programmiersprachen* bereits gegeben sind, sollen an dieser Stelle nochmals Unterschiede speziell zur Konfiguration *technischer* Systeme aufgeführt werden. Diese Unterschiede liefern später wichtige Ansatzpunkte, um die Beschreibungsmittel der Systemkonfigurationen und vor allen Dingen die *Aufgabenkomplexität* für die Schülerinnen und Schüler der Sekundarstufe I erneut sinnvoll zu begrenzen. Folgende Unterschiede lassen sich

[9] Vermutlich sind Schülerinnen und Schülern systolische und diastolische Werte unbekannt. Dies Beispiel soll lediglich die Struktur einer Konfiguration verdeutlichen. Außerdem kann es als Beispiel aufgegriffen werden, wenn in Kapitel 4 das Augenmerk auf fächerübergreifenden Unterricht gelegt wird.

erkennen:

1. *Die Sensoren und Aktoren sind „black boxes“*. Das ist gar nicht anders möglich. Ein Verständnis für die Funktionsweise von Sensoren zu gewinnen, ist für die Lernenden sicherlich von Wert, in einer Unterrichtseinheit Programmieren hingegen zu zeitraubend und nicht im eigentlichen Blickpunkt des Interesses. Ebenso verhält es sich mit der Steuerung von Motoren usw. Man kann nicht die einzelnen Sensoren und Aktoren konfigurieren, sondern nur ihr *Zusammenspiel*. Der Fluss von Eingaben über die Verarbeitung zu Ausgaben ist klar ersichtlich. Diese klare Trennung, die sich bei Techniksystemen ergibt, ist nach Meinung der Autorin didaktisch sehr sinnvoll. Betrachten wir im Gegensatz dazu ein Java-Programm: In einem Dialogfenster können Eingabefelder und Buttons verwendet werden. Diese Eingabefelder liefern durch die Interaktion mit dem Bediener Eingabedaten. Genauso können sie aber auch als Ausgabefeld verwendet werden. Außerdem können im Quellcode Eingabefelder als Objekte selbst konfiguriert werden. Während Sensoren in Techniksystemen nur Eingabewerte liefern und selbst nicht konfiguriert werden können, können Eingabefelder in Java also Eingaben liefern, Ausgaben anzeigen oder selbst Gegenstand einer Konfiguration sein, was für Programmieranfänger sicherlich weniger übersichtlich erscheint.
2. Wie unter Punkt 1 angeklungen, gibt es in Techniksystemen eine *eindeutige* Trennung von Ein- und Ausgabedaten. Sensoren und Aktoren können nicht verwechselt werden. Zwar beeinflussen die Aktionen, welche von den Aktoren ausgelöst werden, durch eine Systemänderung wieder die ermittelten Sensorwerte, als direkte Eingaben können sie jedoch nicht verwendet werden. Das hat den Vorteil, dass folgende didaktische Vereinfachungen in der Struktur der Algorithmen gemacht werden können: *Anweisungen* in den Algorithmen sind immer Ansteuerungen bestimmter Aktoren oder bewirken einen Zustandswechsel des Systems. *Bedingungen*, die bei Alternativen oder auch Schleifen von Bedeutung sind, hängen immer nur von den Sensorwerten oder von Kombinationen der Sensorwerte ab, ggf. noch vom Zustand des Systems.[10] Dadurch wird für die Schülerinnen und Schüler die Struktur von Algorithmen wesentlich vereinfacht und nochmals auf didaktisch sinnvolle Weise reduziert.
3. Charakterisiert man „herkömmliche“ Programmieraufgaben in der Schule (siehe u. a. Engelmann 2007), dann wird in modernen objektorientierten Programmiersprachen, wie z. B. Java oder Delphi, natürlich das Prinzip der Ereignissteuerung verwendet. Eine Oberfläche mit Buttons und Eingabefeldern dient der Erfassung von Eingabedaten. Wird ein

[10] Wie im späteren Verlauf dieses Kapitels gezeigt werden wird, sind auch nur boolesche Variablen nötig.

Ereignis, z. B. Buttonklick, ausgelöst, dann wird in einer mit diesem Ereignis verknüpften Methode ein Algorithmus abgearbeitet, der die Daten in Ausgabedaten zur Ausgabe oder Weiterverarbeitung transformiert. Fast ausschließlich findet man die Verarbeitung von Zeichenketten und Zahlen, ggf. noch booleschen Werten. Wie übertragbar ist aber ein Programm, das auf Buttonklick z. B. Primzahlen berechnet, auf ein technisches System der Lebenswelt der Schülerinnen und Schüler? Vom Grundprinzip der Ereignissteuerung passiert natürlich dasselbe. Die Sensoren ermitteln Werte, die in einem bestimmten vorher festgelegten Wertebereich Aktionen auslösen. Ist aber den Schülerinnen und Schülern der Mittelstufe dieser gedankliche Transfer bewusst? Wenn Lernende nun nur in Java oder Delphi Dialogfenstern Methoden hinzufügen, können sie dann den Abstraktionsschritt nachvollziehen, dass ein Trockner, der bei trockener Wäsche piept, genauso programmiert ist? Die Algorithmik bei der Steuerung technischer Systeme und der Programmierung von Oberflächen ist zwar dieselbe, der Schritt von einer GUI (graphical user interface)-Programmierung zu Systemen der Realität aber zu groß, um das neu gewonnene Wissen auf die Lebenswelt zu übertragen. Vielmehr scheinen die Zielrichtungen hier unterschiedlich zu sein. Konfiguriert man technische Systeme, dann ist Programmieren kein Selbstzweck, sondern dient im Sinne der Allgemeinbildung dem Verständnis der Funktionsweise alltäglicher technischer Systeme.

4. Die Programmierung technischer Systeme der Lebensumwelt hat, wie an späterer Stelle gezeigt werden wird, eine begrenzte Komplexität.

3.4 Algorithmenbausteine zur Konfiguration technischer Systeme

Die Ergebnisse aus Kapitel 2 besagen, dass es zur Beschreibung von Systemkonfiguration einen Satz beliebig kombinierbarer Funktionen geben muss, also elementare Basiselemente, sowie Operationen auf diesen, die beliebig kombiniert werden können und stets lauffähige Konfigurationen des Systems liefern. Sensoren liefern die Eingaben in technische Systeme. Die Werte dieser Sensoren können digital (z. B. bei Schaltern) oder analoge Spannungswerte sein (z. B. bei einem Lichtsensor). Das zu konfigurierende System kann u.U. mehrere Eingänge besitzen. Eine Basisoperation wäre z. B. `ReadInput(`*`Nummer`*`)`, wobei *`Nummer`* ein Eingang sein muss, an dem ein Sensor angeschlossen ist. Die Angabe dieser Basisoperation ändert als alleinige Konfiguration nichts an der Grundfunktionalität des Systems. Das System ist weiterhin in seiner Grundfunktionalität lauffähig. Eine weitere Basisoperation kann `SetOutput(`*`Nummer, Wert`*`)` sein. *`Nummer`* ist in

diesem Fall ein Ausgang, an den ein Aktor angeschlossen ist, und `Wert` entweder ein analoger Spannungswert für diesen Aktor oder auch ein digitaler Wert, je nach verwendetem Aktor. Konfiguriert man ein technisches System, bei dem ein Motor an Ausgang 1 angeschlossen ist, mit dem Befehl `SetOutput(1, 255)`, so ändert sich die Funktionalität des Systems so, dass nun der Motor läuft. Diese beiden Basiselemente bilden die kleinstmögliche Menge von Basisoperationen für diesen Fall. Weitere Operationen nach demselben Schema sind natürlich zulässig.
Welche Operationen oder Mittel zur Kombination auf den Basiselementen sind notwendig und nach der Definition aus Kapitel 2 legitim?
Eine Hintereinanderausführung oder Sequenz der Basiselemente muss möglich sein. Ein Trockner kann schließlich piepen und anschließend auf dem Display eine Information ausgeben.
Eine weitere Operation auf den Basiselementen ergibt sich aus folgendem Beispiel:

Beispiel 14:

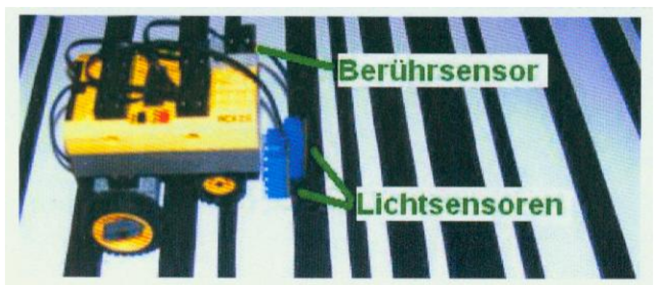


Abbildung 11: Ein von Schülern mit dem LEGO-System konstruierter Strichcodescanner

Im Unterricht bauten Schülerinnen und Schüler einer 7. Klasse dieses vereinfachte Modell eines Strichcodescanners mit LEGO. Das System besteht aus zwei Lichtsensoren, die im Abstand einer Strichbreite voneinander angebracht sind, und einem Berührsensor. Auf dem vergrößerten Strichcode kann der Scanner vier mögliche Codes mit den Kennungen: schwarz-schwarz, schwarz-weiß, weiß-weiß und weiß-schwarz unterscheiden. Als Aktionen wurden vier unterschiedliche Töne gewählt, durch die die vier Codes gekennzeichnet werden.

Das Schülerprogramm ist unten stehend in Abbildung 12 gegeben und zeigt, dass neben den elementaren Operationen die Alternative ein wichtiges Konstrukt ist und damit als Baustein aufgenommen werden muss.

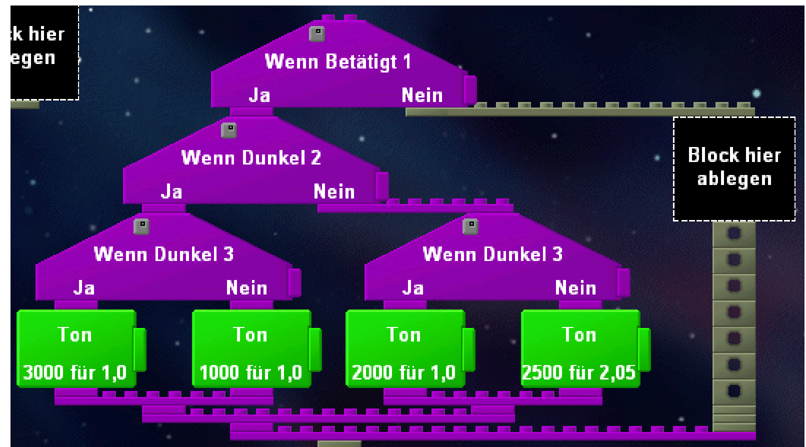


Abbildung 12: Schülerprogramm des Strichcodescanners

Beispiel 15:

Weiter oben wurde bereits die Funktionalität eines Blutdruckmessgerätes erläutert.

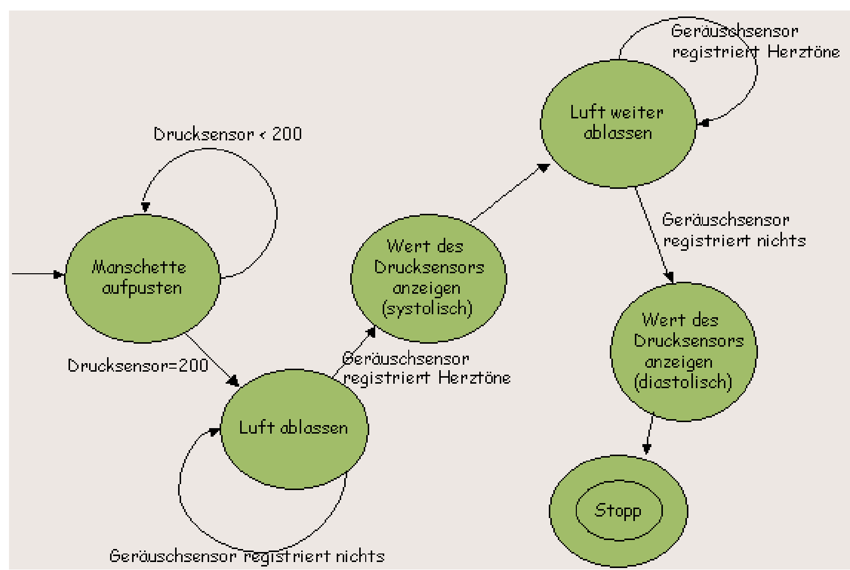


Abbildung 13: Funktionalität eines Blutdruckmessgerätes

Ohne eine Form der Wiederholung können technische Systeme nicht beschrieben werden. Generell arbeiten alle einfachen Regelsysteme, die ja auch technische Systeme darstellen, fortlaufend. Ob allerdings eine Wiederholung iterativ oder rekursiv zu sehen ist, wird an späterer Stelle diskutiert.

Hier wurde durch Beispiele gezeigt, welche Bausteine für die Konfiguration technischer Systeme nötig sind. An späterer Stelle werden diese Operationen auf den Basiselementen formalisiert. Durch die Analogie zu imperativen Programmiersprachen lässt sich aber an dieser Stelle schon erahnen,

dass die genannten Operationen beliebig kombinierbar sind und den Anforderungen an die Beschreibungsmittel einer Systemkonfiguration aus Kapitel 2 genügen.

Schwieriger zu zeigen ist allerdings, warum ein bestimmter Baustein nicht verwendet werden soll. Als Basiselement imperativer Programmiersprachen wurde in Kapitel 2 die Operation „Zuweisung“ angegeben, der das Variablenkonzept zugrunde liegt. Variablen können Werte annehmen und dienen damit der Speicherung von Informationen. Durch die ausschließliche Konfiguration technischer Systeme können hier Einschränkungen gemacht werden, wie der nächste Abschnitt zeigen wird. Diese Einschränkung reduziert die Komplexität der Lerneinheit für die Schülerinnen und Schüler aber, ohne das Ziel zu beschränken, die Lernenden zu befähigen, die Funktionalität technischer Systeme ihrer Lebensumwelt algorithmisch beschreiben zu können.

3.4.1 Steuerungssysteme

Technische Systeme besitzen Sensoren, die permanent Eingaben liefern, und Aktoren, die gemäß der Konfiguration darauf reagieren. Ein einfaches Beispiel ist die Heizungsanlage in der Wohnung. Man stellt eine Wunschtemperatur ein, z. B. 23 °C. Ein Fühler misst die aktuelle Temperatur. Nur wenn der aktuelle Wert kleiner als der Sollwert (23 °C) ist, wird die Heizung eingeschaltet. Solche Systeme werden als „Regelkreise“ bezeichnet. Das Deutsche Institut für Normung schreibt: „Das *Regeln*, die *Regelung*, ist ein Vorgang, bei dem *fortlaufend* eine Größe, die Regelgröße (zu regelnde Größe), erfasst, mit einer anderen Größe, der Führungsgröße, verglichen und im Sinne einer Angleichung an die Führungsgröße beeinflusst wird. Kennzeichen für das Regeln ist der geschlossene Wirkungsablauf, bei dem die Regelgröße im Wirkungsweg des Regelkreises fortlaufend sich selbst beeinflusst“ (DIN 19226, Teil 1).
Was aber ist mit dem Beispiel der Jalousien, die automatisch heruntergelassen werden, wenn ein Lichtsensor anzeigt, dass es draußen dunkel wird? In Wikipedia finden wir: „Ist der fortlaufende Vergleich nicht vorhanden, spricht man von einer *Steuerung*. Eine Heizung, die nur die *Außentemperatur* misst und entsprechend den Raum beheizt, ist eine Steuerung. Das Heizen hat auf die Außentemperatur keinen Einfluss. Es wird also nichts zurückgeführt“ (wikipedia 2008b).
„Die *Steuerungstechnik* umfasst den Entwurf und die Realisierung von Steuerungen. Sie ist ein Teilgebiet der Automatisierungstechnik. Die *Steuerung* beeinflusst den Arbeitsablauf eines Gerätes oder eines Prozesses nach einem vorgegebenen Plan. Abhängig von Eingangsgrößen (Schalter, Zeitpunkt) und Zustandsgrößen (Motor läuft, aktuelle Temperatur) werden Ausgangsgrößen (Motor, Ventil) gesetzt. Im Gegensatz zur Regelung fehlt bei der Steuerung die fortlaufende Rückkopplung der Ausgangsgröße auf den Eingang“ (wikipedia 2008a).

Die Systeme, die in dieser Arbeit konfiguriert werden sollen, können also auch unter dem Begriff der *Steuerungssysteme* zusammengefasst werden. Jetzt stellt sich die Frage, wie Steuerungssysteme in der Realität programmiert werden. Das Schlagwort ist hier die *SPS-Programmierung* (Speicherprogrammierbare Steuerung). „Eine *Speicherprogrammierbare Steuerung* (SPS, engl. *Programmable Logic Controller*, *PLC*) ist eine Baugruppe, die zur Steuerung oder Regelung einer Maschine oder Anlage eingesetzt wird. In der Regel ist eine solche Baugruppe elektronisch ausgeführt und ähnelt den Baugruppen eines Computers. Die Geber (Sensoren) und die Stellglieder (Aktoren) sind mit dieser Baugruppe verbunden. Das zugehörige Betriebssystem (Firmware) stellt sicher, dass dem Anwenderprogramm immer der aktuelle Zustand der Geber zur Verfügung steht. Anhand dieser Informationen kann das Anwenderprogramm die Stellglieder so ein- oder ausschalten, dass die Maschine oder die Anlage in der gewünschten Weise funktioniert“ (wikipedia 2008a).

In DIN EN 61131 sind fünf Programmiersprachen für die SPS-Programmierung spezifiziert:

(i) AWL, vergleichbar mit Assembler

(ii) KOP, vergleichbar mit einem Elektroschaltplan

(iii) FBS oder FUP, ein Funktionsplan (mit Parallelverzweigungen) ähnlich einem Flussdiagramm

(iv) AS, eine Art Zustandsdiagramm

(v) ST, als strukturierter Text an Hochsprachen angelehnt.

Betrachtet man die Sprachen genauer, insbesondere KOP und FUP, dann handelt es sich um Schaltnetze, Schaltwerke, Zustandsgraphen und Ablaufsteuerungen darauf.

Beispiel 16:

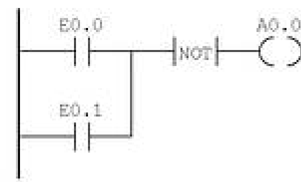


Abbildung 14: KOP-NOR-Verknüpfung (aus SPS-Lehrgang 2009)

Abbildung 15: SR-Speicherglied im KOP (aus SPS-Lehrgang 2009)

Schaltwerke können durch Zustandsgraphen (Mealy-Maschinen[11]) beschrieben werden. Wie Zustandsgraphen (Mealy-Maschinen) in Schaltwerke überführt werden können, kann u. a. in Modrow (Modrow 2004) oder anderen Lehrbüchern (Schiffmann, Schmitz 2003) nachgelesen werden. Zustandsgraphen haben nicht die Möglichkeit eines unbegrenzten Speichers. Lediglich durch eine Änderung der Zustände können gewisse Informationen „gemerkt“ werden. Da endliche Automaten nur Worte regulärer Sprachen erkennen, sind sie nicht universell verwendbar. Es gibt Algorithmen, die nicht mit regulären Sprachen beschrieben werden können, z. B. die Sprache aller möglichen Palindrome[12]. Ein (imperatives) Programm, welches dieselbe Konfiguration beschreibt wie ein endlicher Automat, benötigt demnach als Variablentyp nur boolesche Variablen, die wie „flags“ verwendet werden.

Beispiel 17:
Man verwendet so viele Variablen wie Zustände im Graphen gegeben sind. Der Wert der Variablen, die den Startzustand simuliert, ist `true`, die Werte aller anderen `false`. Jeder Zustandsübergang im Zustandsgraphen wird durch eine (geschachtelte) `if`-Anweisung beschrieben. Bei Zustandsänderung ändern sie die Variablenbelegungen entsprechend.

Wenn Steuerungssysteme, also die technischen Systeme unseres Betrachtungsspektrums, sogar in der Realität in Sprachen programmiert werden, die nur ein eingeschränktes Variablenkonzept beinhalten, z. B. nur boolesche Variablen verwenden, dann genügt dieses Konzept auch für die Konfiguration technischer Systeme in der Mittelstufe eines Gymnasiums.

Zum Abschluss dieses Abschnittes soll ein letzter Algorithmenbaustein zur Konfiguration technischer Systeme betrachtet werden:
Schülerinnen und Schüler, die sich in einer Unterrichtseinheit „Systemkonfiguration“ mit den oben beschriebenen Konzepten und Algorithmenbausteinen auseinandergesetzt haben und Technik-

[11] Mealy-Maschinen sind endlichen Automaten ähnlich, lassen jedoch abhängig von der Eingabe und dem aktuellen Zustand auch Ausgaben zu.
Definition:
$M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$
Q ... endliche Menge von Zuständen
Σ ... Eingabealphabet
Δ ... Ausgabealphabet
λ ... totale Funktion, $\lambda : Q \times \Sigma \rightarrow \Delta$
δ ... totale Überführungsfunktion, $Q \times \Sigma \rightarrow Q$
q_0 ... Startzustand ($q_0 \in Q$)

[12] Ein Palindrom ist ein Wort, welches vorwärts und rückwärts gelesen gleich ist.

systeme der Lebenswelt konfigurierten, haben in einer anschließenden Klausur einen *grundsätzlichen* „Fehler“ begangen. Die Schülerinnen und Schüler hatten ihre selbst entwickelten Algorithmen mit Hilfe von Struktogrammen[13] zu dokumentieren.

In der zur unten stehenden Schülerantwort gehörenden Aufgabe war die Konfiguration eines Roboters mit zwei Lichtsensoren und zwei Motoren gesucht, der einer schwarzen Linie folgen kann.

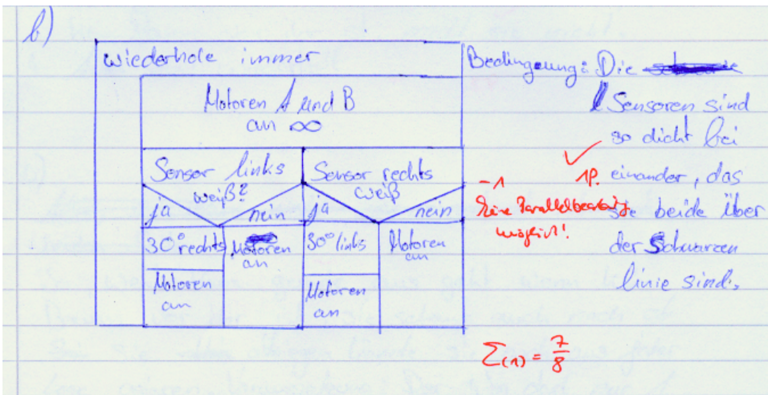


Abbildung 16: Schülerantwort als Struktogramm in einer Klausur

Die Idee dieses Schülers war, dass der Roboter geradeaus fährt und beide Lichtsensoren über der schwarzen Linie hält. Verläuft die Linie nun nicht geradeaus, dann bemerkt einer der beiden Sensoren plötzlich Weiß. In diesem Moment steuert der Roboter dagegen, indem er im Falle, dass der linke Sensor Weiß erkennt, nach rechts und im Falle, dass der rechte Sensor Weiß erkennt, nach links fährt. Mehrere Schülerinnen und Schüler haben Programmteile für beide Sensoren geschrieben und versucht, im Struktogramm eine nicht erlaubte *Parallelität* der beiden Programmteile darzustellen, wie in Abbildung 16 ersichtlich.

In vielen technischen Systemen laufen mehrere Befehlsströme parallel nebeneinander. Zwar handelt es sich nicht um echte parallele Systeme, weil die Teilaufgaben disjunkt voneinander zu betrachten sind, aber z. B. in einem Trockner findet man einen Regelkreis, der die Feuchtigkeit der Wäsche überprüft und daraufhin die Trocknerzeit reguliert, neben einem Regelkreis, der die Temperatur misst und daraufhin das Heizgebläse steuert.

Die Schülerinnen und Schüler brauchen also Modelle für Systeme, die parallele Prozesse zulassen, und das leisten D-Diagramme oder Struktogramme eben nicht.

[13] Verwendet werden im bisherigen Unterricht die sogenannten Nassi-Shneiderman-Diagramme nach DIN 66261 (wikipedia 2009e)

Es stellt sich also die Frage, ob man die Parallelität von Prozessen als Algorithmenbaustein in die Programmierung integrieren muss. „Zwei Prozesse A und B heißen parallel, wenn sie voneinander unabhängig und gleichzeitig ablaufen können“ (Wagenknecht 2004). Probleme bereitet die Parallelität von Anweisungen allerdings in folgenden Fällen:

1. Das zugrunde liegende Modell des Von-Neumann-Rechners als Teil des Systems reicht dann nicht mehr aus. Man müsste andere Modelle, wie z. B. Petri-Netze, in den Unterricht integrieren.
2. Mit der Parallelität von Prozessen und mit der dann nötigen Synchronisation ergeben sich daraus resultierende Phänomene wie Verklemmung, Semaphor, kritischer und unkritischer Abschnitt und andere (Rauber, Rünger 2007). Auch diese Probleme müssen dann im Unterricht der Sekundarstufe I thematisiert werden. Ansätze dazu, die auch zu dem technikorientierten Programmieransatz dieser Arbeit passen, finden sich bei Reichert et al. (Reichert, Nievergelt, Hartmann 2005) in der (graphischen) Programmierumgebung „MultiKara“. Auch Schubert und Schwill (Schubert, Schwill 2004) zeigen anhand von Beispielen, dass sich diese Probleme mit Hilfe von Petri-Netzen im Unterricht veranschaulichen lassen.
3. Auf der anderen Seite spiegelt ein Petri-Netz nicht die Realität eines zugrunde liegenden Systems wieder. Wenn die Systemkonfiguration technischer Systeme durch allgemeinbildende Aspekte wie „Weltorientierung“ legitimiert werden soll, dann sollten auch zugrunde liegende Rechenmodelle der Realität verwendet werden. Und das sind nach wie vor nur scheinbar parallele, intern jedoch sequentielle Verarbeitungssysteme.

Einen Ausweg bietet folgende Möglichkeit:

Was muss bei parallelen Prozessen denn synchronisiert werden? In erster Linie doch der Speicherzugriff. Da in dieser Arbeit jedoch das Variablenkonzept sehr beschränkt ist und Variablen höchstens als „flags“ benutzt werden, steht ein Speicherzugriff, der synchronisiert werden muss, in keiner Relation zum Nutzen. Um dennoch der Vorstellung der Schülerinnen und Schüler gerecht zu werden, dass in ein und demselben System mehrere Steuerungsprozesse gleichzeitig nebeneinander ablaufen, bedient sich diese Arbeit des Begriffs der *disjunkten Parallelität*.

Wir konstatieren:

Zwei einzeln darzustellende Konfigurationen können gleichzeitig ablaufen, und es werden Darstellungsformen gewählt, in der dies visualisiert werden kann. Die Konfigurationen sind aber disjunkt, haben möglicherweise einen gemeinsamen Start- oder Endpunkt, aber keine weiteren Berührungspunkte.

Beispiel 18:

„Wir stellen uns eine Testperson P in einem Raum mit zwei Maschinen M1 und M2 vor. Die Person erstellt auf der Maschine M1 ein Manuskript von ca. 380 Seiten Länge. Um dieses Manuskript zu formatieren, benötigt das Formatierungsprogramm auf M1 ca. 20 Minuten, Zeit genug für P, eine Partie Schach gegen die Maschine M2 zu spielen. Beide Abläufe, Formatierung und Schachspiel, bestehen jeweils aus einer sequentiellen Abfolge von Verarbeitungsschritten, sind aber gegenseitig nicht [...] geordnet; man sagt auch die beiden Abläufe sind disjunkt parallel“ (Best 1995)

3.4.2 Formalisierung

In Anlehnung an die Formalisierung imperativer Programme gemäß Kapitel 2.5.1 werden die Beschreibungsmittel der Konfiguration technischer Systeme, die im vorhergehenden Abschnitt erarbeitet wurden, folgendermaßen formalisiert:

Eine Konfiguration eines technischen Systems ist ein Wort der von *C* erzeugten Sprache.

C::= $C_1 \| C_2$

C_1::= Anweisung

Anweisung ::= Ausgabewerte für Aktor (*Nummer*) auf *Wert* setzen

Anweisung ::= Eingabewert Sensor (*Nummer*) lesen

Anweisung::= Anweisung_1**;** Anweisung_2

Anweisung::= *variable*=**true** | **false** | $\mathit{variable}_1$ **and** $\mathit{variable}_2$ | $\mathit{variable}_1$ **or** $\mathit{variable}_2$ | **not** $\mathit{variable}_1$

Anweisung::= **if** Bedingung_1 → Anweisung_1 □ ... □ Bedingung_n → Anweisung_n **fi**

Anweisung::= **do** Bedingung_1 → Anweisung_1 **od**

Bedingung::= variable | Eingabewert Sensor(*Nummer*) lesen < W*ert* | …>... | ...=...

Kontextbedingungen:

(i) *Nummer* und *Wert* sind gültige natürliche Zahlen.

(ii) *Variablen* sind mit true vorbelegt und haben einen eindeutigen Namen.

(iii) Die Alternative lässt keinen Nichtdeterminismus zu.[14]

Die Semantik dieser Bausteine ergibt sich intuitiv, und es wird auf die Angabe einer formalen Semantik verzichtet. (Mit $C_1 \| C_2$ sind die disjunkten Konfigurationen C_1 und C_2 gemeint.)

[14] Bei Steuerungssystemen und Algorithmen in der Sekundarstufe I ist das Konzept nichtdeterministischer Programme unnötig, bei Steuerungssystemen sogar gefährlich. Man denke nur an ein Beatmungsgerät, welches nichtdeterministisch reagiert.

Mit diesen Angaben, in Verbindung mit Kapitel 2.5.1, sind die Funktionen identifiziert und semi-formal beschrieben, die notwendig sind, um Techniksysteme zu konfigurieren.

In Kapitel 2.5.1 wurden die Definitions- und Wertemenge der Funktionen durch Zustände gegeben. Übertragen bedeutet ein Zustand in diesem Fall die aktuelle Belegung aller Variablen sowie die aktuellen Werte an allen Eingängen und allen Ausgängen des Systems. Damit erschließt sich auch, dass dieser Satz von Basiselementen und Operationen den Anforderungen der Definition einer Systemkonfiguration genügt.

Die Aufgabe, die bestehen bleibt, ist eine Form der Codierung zu finden, die semantisch mit den oben beschriebenen Funktionen übereinstimmt, aber eine Form hat, die für Schülerinnen und Schüler der Sekundarstufe I angemessen ist.

3.5 Konfigurationssprachen technischer Systeme

„Das wichtigste Werkzeug des Programmierers bei der Spezifikation von Informationsverarbeitungsprozessen ist die Programmiersprache. Da diese die Ausdrucksmittel bestimmt, die dem Programmierer bei der Lösung von Problemen zur Verfügung stehen, ist ihre Auswahl von großer Bedeutung“ (Klaeren, Sperber 2007).
In Kapitel 2 wurde kurz auf graphische Programmierumgebungen eingegangen oder solche, die Syntaxfehler der Schülerinnen und Schüler begrenzen.

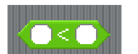
3.5.1 Graphische oder textbasierte Programmierung?

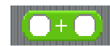
Beobachtet man die Schülerinnen und Schüler im Umgang mit universellen Programmiersprachen wie Java oder Delphi, dann sind ein Großteil der Fehler, die ihnen während des Programmierens unterlaufen, syntaktischer Natur. Gerade diese Fehler und die mühsame Fehlersuche z. B. des fehlenden Semikolons tragen aber nicht zum besseren Verständnis der zugrunde liegenden Lösungsstrategie bei. Es sind lediglich Rechtschreibfehler bei der Formulierung einer (eventuell) richtigen Idee. Einige Tools ermöglichen durch sogenannte graphische Programmierung einen Verzicht auf das Erlernen einer textbasierten Syntax. In diesen graphischen Sprachen können Elemente nur an „korrekten“ Stellen platziert werden.

Beispiel 19:

In der graphischen Programmierumgebung „scratch“ (scratch 2009) werden boolesche und arithmetische Ausdrücke unterschieden. Zahlen stehen dabei in abgerundeten Kästen, boolesche Werte in Kästen mit „Spitzen“. In einer Alternative können als Bedingung nur Kästen mit „Spitzen“ eingefügt werden.

Boolesche Werte: Arithmetische Ausdrücke: Alternative:





Trotzdem kann man (wie später gezeigt werden wird) die komplette Algorithmik damit unterrichten. Kann also im Rahmen einer Stärkung des Selbstwertgefühls der Schülerinnen und Schüler und diese Stärkung geht von lauffähigen Programmen aus, die die gegebene Problemstellung lösen darauf verzichtet werden, sie durch das mühsame Erlernen der Syntax und das Aufspüren von Syntaxfehlern zu frustrieren?

Zur Klärung dieser Frage soll ein Vergleich aus der Mathematikdidaktik herangezogen werden. Das Finden einer Transformation, die Eingabewerte auf Ausgabewerte abbildet, das Finden eines Algorithmus oder einer Konfiguration, kann man mit einer Beweisaufgabe im Mathematikunterricht vergleichen. In Kapitel 4 wird dies näher erläutert werden. Dieser Vergleich ergibt sich aus den Arbeiten von Regina Bruder (Bruder, Leuders, Büchter 2008).

Mathematische Beweise können wie an der Universität in formaler Schreibweise dargestellt werden. Diese formale (mathematisch korrekte) Schreibweise ist aber offensichtlich nicht für Schülerinnen und Schüler der Sekundarstufe I geeignet, obwohl auch dort Beweise durchgeführt werden, die meist anschaulicher Art sind, wie Beispiel 20 auf der folgenden Seite zeigt.

In den von der Kultusministerkonferenz veröffentlichten Bildungsstandards Mathematik (Bildungsstandards 2003) findet sich die prozessbezogene Kompetenz „mathematisch argumentieren“. Unter dem Stichwort „Beweisen“ oder „Begründen“ ist oft ein umgangssprachliches oder anschauliches Begründen zu verstehen. Auf eine streng formale Schreibweise kommt es viel weniger an als auf das Verständnis für den Inhalt. Die Schulmathematik vertritt an dieser Stelle also den Standpunkt, dass Mathematik verstanden und auch angewendet werden kann, ohne dass es einer formalen Schreibweise bedarf. Es stellt sich entsprechend für die Informatik die Frage, wie genau und formal korrekt Konfigurationen aufgeschrieben werden müssen. Analog zur Schulmathematik kann man nach Meinung der Autorin Algorithmik unterrichten, Algorithmen

entwickeln und nachvollziehen, ohne gezwungen zu werden, sie textbasiert fehlerfrei zu codieren.

Beispiel 20:

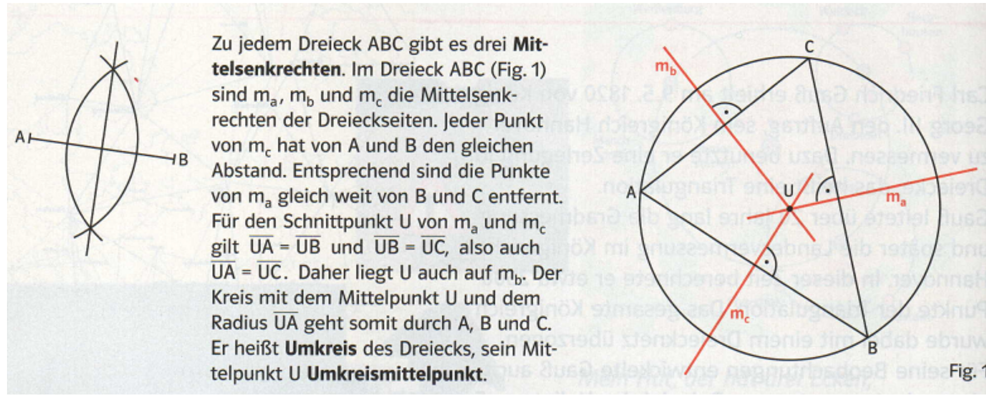

Zu jedem Dreieck ABC gibt es drei **Mittelsenkrechten**. Im Dreieck ABC (Fig. 1) sind m_a, m_b und m_c die Mittelsenkrechten der Dreieckseiten. Jeder Punkt von m_c hat von A und B den gleichen Abstand. Entsprechend sind die Punkte von m_a gleich weit von B und C entfernt. Für den Schnittpunkt U von m_a und m_c gilt $\overline{UA} = \overline{UB}$ und $\overline{UB} = \overline{UC}$, also auch $\overline{UA} = \overline{UC}$. Daher liegt U auch auf m_b. Der Kreis mit dem Mittelpunkt U und dem Radius $\overline{UA}$ geht somit durch A, B und C. Er heißt **Umkreis** des Dreiecks, sein Mittelpunkt U **Umkreismittelpunkt**.

Abbildung 17: „Beweis“ in einem Mathematikschulbuch einer 8. Klasse (Lambacher Schweizer 2007)

In der Arbeit von Reichert et al. (Reichert, Nievergelt, Hartmann 2005) mit Bezug auf die Arbeiten (Boulay et al. 1999), (Brusilovsky et al. 1997) werden die Anforderungen an ein Programmiertool für Einsteiger benannt. Reichert et al. nennen *„Kleiner Sprachumfang. [...] Einfache Programmierumgebung“,* wobei sie schreiben: „Ein einfach zu bedienender Programmeditor hilft Syntaxfehler zu vermeiden und erleichtert die Konzentration auf die Programmlogik“. Unterstützt wird diese These auch von Hendrik Büdding. Er schreibt: „In der Einstiegsphase im Informatikunterricht hat ein visuelles Programmiersystem den Vorteil, dass sich die Fehlerarten und -zahlen reduzieren lassen“ (Büdding 2007). Auch die Unterrichtserfahrung der Autorin lehrt dies, wie folgendes Beispiel zeigt:

Beispiel 21:

Alle Schülerinnen und Schüler einer 8. Klasse, denen die Grundbausteine von Algorithmen durch andere Zusammenhänge vertraut waren, haben sich die graphische Programmiersprache „scratch“ (scratch 2009) ohne Hilfe des erkrankten Lehrers während der Vertretungsstunden bei einem fachfremden Lehrer angeeignet. Offene Fragen und Probleme wurden in der Gruppe geklärt. Nach vier Stunden hatten einige Schülerinnen und Schüler ein Spiel ähnlich dem Klassiker „pong“ selbstständig programmiert. Bei diesem Spiel steuert man mit den Pfeiltasten den roten „Schläger“ nach oben und unten, wobei man versuchen muss, den blauen „Ball“ zu treffen, der vom „Schläger“ und den drei Seiten ohne „Schläger“ abprallt.

In einer 11. Klasse wurde das Spiel „pong“ ebenfalls programmiert, allerdings in Java. Erst nach sechs Wochen (12 Stunden) Unterricht in diesem Themengebiet und mit Hilfestellungen gelang den

Schülerinnen und Schülern eine lauffähige Version.

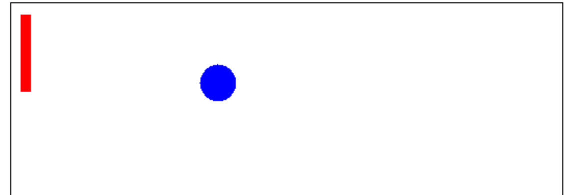


Abbildung 18: screenshot des Spiels "pong"

Graphische Programmierumgebungen haben neben den oben genannten Vorteilen der leichten Erlernbarkeit auch den Vorteil, den kompositionellen Algorithmenentwurf zu unterstützen. Wie in Kapitel 2 beschrieben, sollen Konfigurationen nach dem Baukastenprinzip aus Basiselementen und Operationen darauf zusammengesetzt werden. Dieses Baukastenprinzip wird in der graphischen Programmierumgebung „scratch“ (scratch 2009) ebenfalls gut visualisiert, wie Beispiel 22 zeigt. Ein Vorteil liegt darin, dass neben dem gewünschten analytischen Zugang auch das Experimentieren eine Lösungssuche mittels „Versuch und Irrtum“ darstellen kann.

Beispiel 22:
Auf der folgenden Seite sind zum Vergleich Algorithmen in Java und „scratch“ angegeben.
Gegeben sei der Java-Code eines Programmteils, das für eine gegebene Zahl überprüft, ob es sich um eine *perfekte Zahl* handelt. Dieser Code ist übersichtlich strukturiert. Absichtlich wurde ein Beispiel verwendet, welches für die Anwendung von „scratch“ nicht sonderlich geeignet ist. Derselbe Algorithmus ist nebenstehend in „scratch“ gegeben.
Man kann sehen, dass die Struktur in „scratch“ mindestens genau so übersichtlich ist wie in Java. Farblich lassen sich Kontrollstrukturen von Anweisungen unterscheiden, das Baukastenprinzip wird sichtbar.

```
int zahl =28;
int i=1;
int teilersumme =0;
while (i<zahl){
  if (zahl % i == 0){
    teilersumme=teilersumme+i;
  };
  i++;
};
if (teilersumme == zahl) {
  System.out.println("perfekte Zahl");
}
```

Abbildung 20: Java-Programm „perfekte Zahl“

```
Wenn [Flagge] angeklickt
setze zahl auf 28
setze teilersumme auf 0
setze i auf 1
wiederhole bis i = zahl
  falls zahl mod i = 0
    setze teilersumme auf teilersumme + i
  ändere i um 1
falls teilersumme = zahl
  sage perfekte Zahl!
```

Abbildung 19: „scratch“-Programm „perfekte Zahl“

Natürlich stoßen graphische Programmierumgebungen wie „scratch“ schnell an ihre Grenzen, wenn z. B. abstrakte Datentypen implementiert werden sollen. Wenn man jedoch die Funktionen aus Kapitel 3.4.1 betrachtet, aus denen Konfigurationen technischer Systeme zusammengesetzt sind, lässt sich erschließen, dass graphische Programmiersprachen dafür *ausreichend* sind. Im Folgenden soll dieser Aspekt näher erläutert werden. In dieser Arbeit werden zwei bestehende und eine von der Autorin entwickelte Lernumgebung vorgestellt, bei denen es sich um graphische Programmierumgebungen handelt, die m. E. die Funktionen beinhalten, die zur Konfiguration technischer Systeme notwendig sind.

Ein zweiter Auswahlgrund für diese drei Lernumgebungen ist, wie in Kapitel 6 näher erläutert wird, die Vielschichtigkeit der Notationsform, die sich durch alle Gebiete der Informatik zieht.

Im Zusammenhang mit der Betrachtung verschiedener Programmierparadigmen werden die beiden graphischen Programmierumgebungen Automaten-Kara (Reichert, Nievergelt, Hartmann 2005) und LEGO-RIS (Robotic Invention System 2.0) (LegoRIS 2009) und ihr Bezug zur Konfiguration technischer Systeme im folgenden Abschnitt vorgestellt.

Um die Funktionen, die Eingabewerte auf Ausgabewerte transformieren, besser identifizieren zu können, soll an dieser Stelle der Begriff des Zustands nach Kapitel 2.5.1 für alle Beschreibungsmittel erweitert werden. Um das Gewicht nicht zu sehr auf imperative Sprachen zu legen, soll ein Zustand nicht nur die aktuellen Eingangs- und Ausgangswerte umfassen, sondern auch den aktuellen „Zustand“ des Systems (analog zur aktuellen Variablenbelegung).

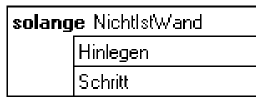
Um begrifflich Unterschiede machen zu können, wird an dieser Stelle der Begriff des *Szenarios* als Definitions- und Wertemenge der Funktionen einer Konfiguration von der Autorin eingeführt.
Als *Szenario* definiert diese Arbeit eine Art Schnappschuss des Systems zu einer bestimmten Zeit. In ein Szenario eingeschlossen sind die aktuellen Werte der Sensoren, die aktuelle Position des Systems in seiner Umgebung und die aktuellen Zustände der Aktoren.

Beispiel 23:
Betrachten wir den Roboter Karol (Karol 2009). Karol hat u. a. einen Sensor, der erfassen kann, ob eine Wand vor ihm ist. Karols Welt ist von Wänden eingeschlossen. Die Aktionen, die Karol ausführen kann, sind u. a. „einen Schritt vorwärts gehen" oder „einen Ziegelstein vor sich ablegen". Der Zusammenhang zu Motoren, die diese Bewegungen ausführen und je nach Konfiguration angesteuert werden, erschließt sich von selbst. Das aktuelle Aussehen von Karols Welt (die Anordnung der Wände und Ziegel), Karols Position und Blickrichtung und die Werte seiner Sensoren bilden ein Szenario.
Gegeben sei folgendes Programm:

```
solange NichtistWand tue
   Hinlegen; Schritt
*solange
```

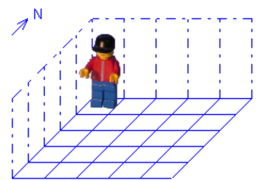
beziehungsweise:

Hauptprogramm

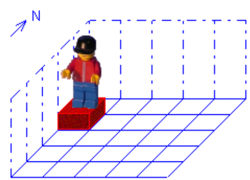


Dann gilt:

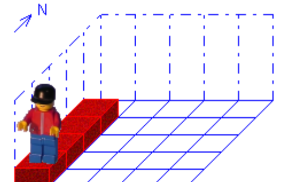
Anfangsszenario *vor* der Ausführung des Programms:



Szenario nach *einem* Schleifendurchlauf:



Szenario am *Ende* des Programms:



3.5.2 Verschiedene „Programmierparadigmen“ in der Konfiguration technischer Systeme

Nachdem in Kapitel 3.4.1 geklärt wurde, welche Bausteine zur Konfiguration verwendet werden sollen, stellt sich die Frage, in welchem Programmierparadigma die Schülerinnen und Schüler ihre Algorithmen implementieren sollen. „Verschiedene Menschen haben unterschiedliche Sichtweisen bei der Beschreibung und Lösung von Problemen. Das hat sich auch in der Welt der Programmiersprachen durch verschiedene Programmierparadigmen manifestiert. Ein Programmierparadigma ist eine Sichtweise, die zur Lösung eines Problems mittels einer Programmiersprache eingenommen wird“ (Pepper, Hofstedt 2006).
Da wir es in der Schule mit den verschiedensten Charakteren zu tun haben und wir ein und denselben Lerninhalt auf unterschiedlichsten Wegen vermitteln sollten, um jedem Lernenden die Chance zum Erreichen der Lernziele zu geben, stellt sich zunächst die Frage, ob man die Schülerinnen und Schüler überhaupt zu einem bestimmten Paradigma führen muss. Für jeden Lernenden ist möglicherweise ein anderes Paradigma, eine andere Sichtweise geeignet.
Folgendes Zitat beschreibt die momentan vorhandenen Paradigmen: „Neben der klassischen Einteilung in deklarative und imperative Sprachen sind [...] die Paradigmen der objektorientierten, nebenläufigen und verteilten Programmierung zu sehen. [...] Diese Klassifikation verhält sich orthogonal zur vorher genannten, d. h. es existieren praktisch von allen Sprachparadigmen sowohl sequentielle als auch nebenläufige Vertreter, oft auch verteilte. Und ähnliches gilt auch für die objektorientierte Programmierung: Sowohl für imperative als auch für deklarative Programmiersprachen existieren entsprechende Erweiterungen“ (Pepper, Hofstedt 2006).
Aus rein fachlicher Argumentation wird an dieser Stelle das logische Programmierparadigma ausgeschlossen. Die spezifischen Vorteile und Anwendungsgebiete logischer Sprachen liegen in der „Eleganz und Ausdrucksstärke der Möglichkeiten, mit Funktionsinvertierung und unvollständigen Daten zu arbeiten, sowie spezielle Suchstrategien anzuwenden“ (Pepper, Hofstedt 2006). „Das Grundprinzip der logikbasierten Programmierung besteht darin, eine Datenbasis anzulegen und Fragen zu formulieren, die mit genau dieser Datenbasis und einer eingebauten Suchstrategie beantwortet werden können“ (Wagenknecht 2004). Ein Vorteil, der an dieser Stelle nicht genutzt werden kann. Es gibt bei Systemkonfigurationen keine Datenbasis oder unvollständigen Daten, sondern es muss konkret angegeben werden, wie die gerade vorliegenden Daten, die aktuellen Werte der Sensoren, verarbeitet werden sollen. Um also eine Systemkonfiguration eines technischen Systems zu beschreiben, bleiben die Paradigmenansätze der imperativen und funktionalen Programmierung.

3.5.2.1 Systemkonfiguration und funktionales Paradigma

„Die Funktionale Programmierung basiert auf dem Funktionsbegriff der Mathematik. Eine Funktion bildet Eingabewerte, d. h. Elemente aus dem Definitionsbereich, eindeutig auf Ausgabewerte, d. h. Elemente aus dem Wertebereich, ab" (Pepper, Hofstedt 2006). Dabei *operieren Funktionen auf Daten und können ineinander geschachtelt sein*. Ein wichtiges Prinzip der Informatik, welches in der Funktionalen Programmierung konsequent angewendet wird, ist das Prinzip der Rekursion. Vom Grundgedanken her erlauben funktionale Programme keine Zuweisungen, weshalb keine Nebeneffekte auftreten können. Weiterhin ist eine *Nebenläufigkeit* leicht zu integrieren. Die Grundstruktur einer Funktion wird in Präfixschreibweise notiert und lautet:[15]

(Operation Operand_1 … Operand_n)

Beispiel 24:

(+ 7 3) liefert 10

Beispiel 25:

```
(define factorial
  (lambda (n)
    (if (= n 0) 1
       (* n (factorial(- n 1)))))) 
```
liefert n!.

In Kapitel 2 wurde bereits angegeben, dass funktionale Programmiersprachen orthogonalisiert sind in dem Sinne, dass Funktionen beliebig kombiniert werden können und immer wieder gültige Programme liefern. In diesem Sinne erfüllen funktionale Sprachen also die Anforderung an Beschreibungsmittel der Systemkonfiguration. Funktionale Programme werden nach dem Baukastenprinzip zusammengesetzt, und dies entspricht dem kompositionellen Entwurf, der auch in der Definition der Systemkonfiguration gefordert wurde. Funktionale Sprachen bauen wie die Systemkonfiguration darauf auf, die Transformationen zwischen Eingabedaten und Ausgabedaten als Funktion zu beschreiben. Die Konfiguration technischer Systeme hat gezeigt, dass nach Möglichkeit eine Nebenläufigkeit disjunkt paralleler Prozesse erlaubt sein muss, was sich im funktionalen Paradigma gut integrieren lässt. Im funktionalen Paradigma entfällt zwar weitgehend die Zuweisung, was aber kein Hindernis darstellt, da das Variablenkonzept bei Systemkonfigurationen technischer Systeme auch auf boolesche Variablen („flags") eingeschränkt ist.

Am Beispiel Automaten-Kara und später in Kapitel 5 wird gezeigt, dass der Grundgedanke funktionaler Programmierung in graphischen Programmierumgebungen zur Konfiguration

[15] Hier wird die funktionale Programmiersprache Scheme verwendet.

technischer Systeme sinnvoll verwendet werden kann.

Automaten-Kara

Kara ist eine an der ETH Zürich entwickelte Programmierumgebung für Einsteiger (siehe dazu Reichert, Nievergelt, Hartmann 2005). Kara ist ein Marienkäfer, der in einer Welt lebt, in der es unbewegliche Baumstümpfe, verschiebbare Pilze und Kleeblätter gibt. Kara besitzt fünf Sensoren und kann fünf verschiedene Aktionen ausführen.

Sensoren		**Aktionen**	
Kara erkennt, ob sie vor einem Baum steht. Der Sensor liefert die Werte „yes" oder „no"		Kara geht ein Feld in Blickrichtung vorwärts.	
Kara erkennt, ob sie auf einem Kleeblatt steht. Der Sensor liefert die Werte „yes" oder „no"		Kara dreht sich um 90° nach rechts.	
Kara erkennt, ob rechts neben ihr ein Baum steht. Der Sensor liefert die Werte „yes" oder „no"		Kara dreht sich um 90° nach links.	
Kara erkennt, ob links neben ihr ein Baum steht. Der Sensor liefert die Werte „yes" oder „no"		Kara hebt ein Kleeblatt unter sich auf.	
Kara erkennt, ob sie vor einem Pilz steht. Der Sensor liefert die Werte „yes" oder „no"		Kara legt ein Kleeblatt unter sich ab.	

Tabelle 3: Karas Sensoren und Aktionen

Zunächst wird eine spezielle Kara-Welt angegeben, also ein Anfangsszenario und eine Konfigurationsaufgabe für dieses Systems:

Beispiel 26:

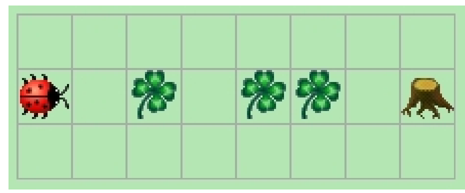


Abbildung 21: Anfangsszenario

Aufgabe*:*

Kara soll bis zum Baum laufen und dort stehen bleiben. Auf ihrem Weg soll sie alle Kleeblätter aufnehmen.

Was hat ein Marienkäfer mit einem technischen System zu tun? In der Tat ist ein Kleeblätter sammelnder Marienkäfer kein Techniksystem aus der Lebenswelt der Schülerinnen und Schüler. (Unter anderem aus diesem Grund hat sich die Autorin für die Entwicklung einer eigenen Lernumgebung entschieden.) Da aber keine geeignetere Lernumgebung zur Verfügung stand, ist im Unterricht Automaten-Kara verwendet worden. Trotz gewisser Abstriche besitzt Kara alle anderen Voraussetzungen für die Darstellung von Systemkonfigurationen. Kara besitzt Sensoren, die permanent Eingaben liefern. Diese Sensoren können abgefragt werden. Kara kann einzelne Aktionen ausführen. Man könnte sich hier die Ansteuerung von Motoren denken, die diese Aktionen ausführen. Die Basiselemente sind auf einige wenige beschränkt. Der weitere Vorteil liegt in der Programmierung des Marienkäfers.

Wie würde Kara in obiger Welt in einer klassischen funktionalen Programmiersprache wie z. B. Scheme programmiert werden?

Man müsste das Problem verallgemeinern zu folgender Aufgabenstellung: Gegeben ist eine Liste mit dem Inhaltstyp {0,1,2}, wobei die Null für ein freies Feld steht, die Eins für ein Feld mit Kleeblatt und die Zwei für einen Baum. Aus der Liste sollen beginnend mit dem Anfang alle Einsen durch Nullen ersetzt werden, bis eine Zwei erscheint. Der Rest der Liste nach der Zwei bleibt unverändert. In „Scheme" (Klaeren, Sperber 2007) definiert man dazu eine Funktion „aufsammeln" wie folgt:

Beispiel 27:

```
(define aufsammeln
  (lambda (ls)
    (cond
      [(eq? (car ls) 1) (cons 0 (aufsammeln (cdr ls)))]
      [(eq? (car ls) 2) (cons 2 (cdr ls))]
      [else
       (cons (car ls) (aufsammeln (cdr ls)))])))
```

Als Listenoperationen dienen car, cdr und cons. car liefert das erste Element der Liste, cdr den Rest. cons kreiert eine neue Liste aus einem Element und einer Liste. cond leitet eine Alternative ein, hier mit den Möglichkeiten: 1. Element = 1, 1. Element = 2, sonst

Ein Aufruf der Funktion in der abstrahierten Welt aus Beispiel 26 liefert:

```
> (aufsammeln '(0 0 1 0 1 1 0 2 ))
(list 0 0 0 0 0 0 0 2)
```

Endszenario

Das Problem demonstriert die Art funktionaler Programmierung, ist aber natürlich ein wenig angepasst, denn es wurden keine Sensoren und Aktoren verwendet.

Kara wird in der Schule mittels endlicher Automaten graphisch programmiert. Das oben genannte Problem, alle Blätter bis zum Baum aufzusammeln, wird folgendermaßen gelöst:

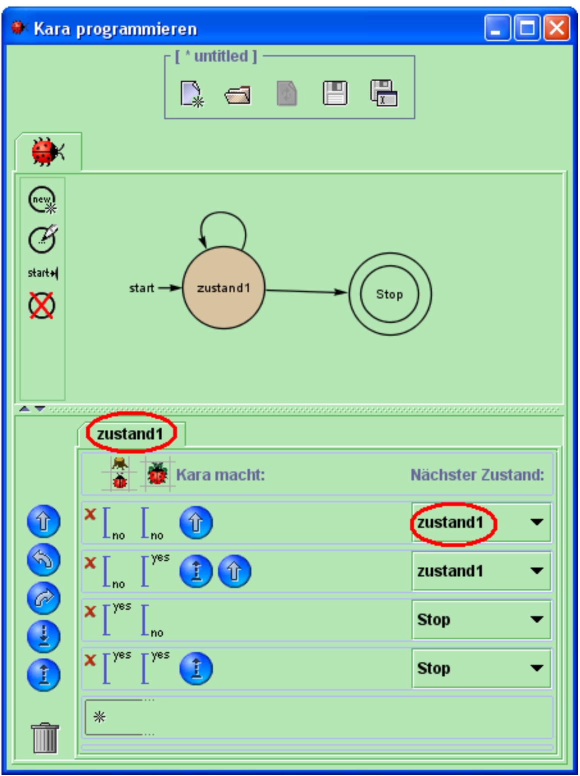


Abbildung 22: Kara-Programm

Die Zustandsübergänge sind unbeschriftet. In dem unten angegebenen Block werden sie jedoch näher spezifiziert. Die Konfiguration des Systems gemäß der Aufgabe aus Beispiel 26 ist eine Funktion, wobei Definitions- und Wertemenge die Menge der möglichen Szenarien sind. Das Anfangsszenario ist in Beispiel 26 gegeben. Es ist intuitiv erfassbar, dass das Endszenario Kara auf einem Feld ohne Kleeblatt direkt vor einem Baum beinhaltet oder (in einem Anfangsszenario ohne Baum) das Programm nicht terminiert.

Betrachtet man den graphischen Block etwas näher, kann man eine partielle Funktion *zustand1* identifizieren, die *rekursiv* definiert ist. Die Funktion *zustand1* ruft (zumindest in zwei der vier möglichen Fälle) wiederum die Funktion *zustand1* auf. Ist s ein Element aus der Menge der möglichen Szenarien, dann beschreibt doch der oben angegebene Block graphisch folgendes:

$$f_{\text{zustand1}}(s) = \begin{cases} f_{\text{zustand1}}(f_{\text{schritt}}(s)) & \text{wenn } \overline{\text{BaumVorne}} \wedge \overline{\text{KleeblattUnten}} \\ f_{\text{zustand1}}(f_{\text{schritt}}(f_{\text{aufheben}}(s))) & \text{wenn } \overline{\text{BaumVorne}} \wedge \text{KleeblattUnten} \\ s & \text{wenn } \text{BaumVorne} \wedge \overline{\text{KleeblattUnten}} \\ f_{\text{aufheben}}(s) & \text{wenn } \text{BaumVorne} \wedge \text{KleeblattUnten} \end{cases}$$

Man kann diese Form der graphischen Programmierung also verwenden, um einen funktionalen Programmieransatz in die Systemkonfiguration zu integrieren. Es gibt elementare Aktionen (wie z. B. Kleeblatt aufheben, die eine Ausgabefunktion darstellen). Bedingungen sind abhängig von den booleschen Werten der Sensoren. Partiell definierte Funktionen sind semantisch äquivalent zu Alternativen und rekursiv definierte Funktionen semantisch äquivalent zu Wiederholungen.
Da technische Systeme der Lebenswelt mit Zustandsgraphen beschrieben werden können, reichen die Möglichkeiten endlicher Automaten auch für die Konfiguration technischer Systeme aus.
Die Forderung der disjunkten Parallelität von in sich sequentiellen Programmen kommt hier nicht zum Tragen. Parallele Automaten mit jeweils eigenem Startzustand, die vollkommen unabhängig voneinander agieren, sind aber denkbar.

Damit ist gezeigt, dass die Grundidee der Programmierung in Kara auch für die Konfiguration technischer Systeme angemessen ist. Statt Sensoren und Aktionen eines Marienkäfers müsste man lediglich andere Sensoren und Aktoren verwenden, am besten ein reales System, das über eine Computerschnittstelle verfügt. Auch wurde gezeigt, dass der Ansatz des funktionalen Paradigmas m. E. in eine graphische Umgebung transportiert werden kann.

Diese Arbeit beschreibt einen Ansatz zum Einstieg in die Programmierung für alle Schülerinnen und Schüler der Sekundarstufe I. Der Ansatz ist in sich geschlossen. Bleiben die Schülerinnen und Schüler jedoch der Informatik auch in der Oberstufe treu, lohnt es sich näher zu untersuchen, ob dieser funktionale Programmieransatz wie hier beschrieben z. B. auf textbasierte Programmiersprachen erweiterbar ist. In Kapitel 6 wird ein Ansatz beschrieben, wie der Übergang von graphischen zu textbasierten Sprachen in einer bestimmten Lernumgebung gelingen kann.
Programmieren die Schülerinnen und Schüler textbasiert funktional und wollen technische Systeme beschreiben, dann stoßen sie in zwei Bereichen an die Grenzen des Paradigmas: Die Zeit sowie eine Interaktion mit dem Bediener mit den daraus resultierenden „unendlichen“ Datenströmen sind schwer integrierbar. Konkret heißt das:

- „In der Funktionalen Programmierung möchte man die Zeit am liebsten ganz loswerden, denn Funktionen sind im wahrsten Sinne des Wortes zeitlos: Der Wert von $\sin(\pi)$ sollte zu Weihnachten kein anderer sein als zu Ostern. Aber spätestens, wenn Interaktionen mit der

Umwelt nötig werden – sei es mit Benutzern am Bildschirm oder mit Sensoren und Aktuatoren in einer Prozesssteuerung –, kann man die Zeit nicht mehr ignorieren. Denn die Welt lebt in der Zeit. Damit müssen [...] auch die funktionalen Programmierer irgendwie mit dem Phänomen Zeit umgehen“ (Pepper, Hofstedt 2006). Weiter schreiben Pepper und Hofstedt: „Diese Frage ist noch immer ein offenes Forschungsthema.“ Es gibt also keine im Paradigma stimmige Möglichkeit der Interaktion. Ohne das Thema an dieser Stelle weiter zu vertiefen, zeigt es uns doch, dass wir den funktionalen Ansatz in reduzierten Aufgaben nutzen können, im Verlauf des Programmierunterrichts aber auf andere Paradigmen ausweichen müssen, wenn komplexere Systeme konfiguriert werden sollen.

- „Bei einem Algorithmus setzt man voraus, dass seine Eingaben zu Beginn der Berechnung bekannt sind und am Ende die Ausgaben geliefert werden. Bei einem reaktiven System wird ein Strom von Eingaben, die nur schrittweise bekannt werden, zu einem Strom von Ausgaben verarbeitet. Es kann mehrere Ein- und Ausgabeströme geben. Die Länge der Ströme ist unbeschränkt. Die Eingaben können von zuvor berechneten Ausgaben abhängen“ (Pomberger, Rechenberg 2002). Die Eingabeströme, die fortlaufend als Sensorwerte in das System fließen, werden von den Schülerinnen und Schülern mit Hilfe von Kombinationen elementarer Operationen so verknüpft, dass das System mit seinen Aktionen in gewünschter Weise auf die Eingabeströme reagiert. Diese Datenströme sind aber unendlich. In der funktionalen Programmierung werden Datenströme als verzögerte Listen dargestellt, die allerdings wiederum ihre ganz eigenen Probleme aufwerfen und nach Ansicht der Autorin für die Schülerinnen und Schüler der Sekundarstufe I noch zu abstrakt und komplex sind. Hier eignen sich Programmierparadigmen, die dieses Problem „eleganter“ lösen können, besser.

Da Nachteile angesprochen wurden, sollen der Vollständigkeit halber auch die Vorteile funktionaler Programmierung für die Schülerinnen und Schüler genannt werden:

- Programmierern der funktionalen Programmierung fällt es aufgrund der strukturierten Denkweise leichter, auf andere Paradigmen umzusteigen (siehe dazu Wagenknecht 2004).
- Funktionale Sprachen sind deskriptive Sprachen. Es gibt eine gute Einbindung von SQL (Structured Query Language), mit der Schülerinnen und Schüler im Informatikunterricht sicherlich in Berührung kommen werden.
- Funktionale Sprachen haben keine Seiteneffekte, unter anderem wegen des fehlenden Prinzips der Zuweisung.

3.5.2.2 Systemkonfiguration und imperatives Paradigma

„In der imperativen Programmierung wird eine Berechnung als Folge von Zustandsübergängen der Variablen betrachtet. Solche Zustandsübergänge können sehr komplex sein und entsprechen dann umfangreichen Prozeduren. Sie können unter Bedingungen stehen oder, mit jeweils anderen Variablenwerten, in Schleifen ausgeführt werden“ (Pomberger, Rechenberg 2002).

Die in Kapitel 3.4.1 gegebene Struktur der Bausteine zur Beschreibung von Konfigurationen technischer Systeme entspricht äußerlich am ehesten dem imperativen Ansatz. Allerdings müssen Zustände als Menge möglicher Szenarien und Zustandsübergänge als Änderungen des Szenarios aufgefasst werden.

Der folgende Abschnitt betrachtet den Einsatz graphischer imperativer Programmierwerkzeuge im Unterricht der Sekundarstufe I am Beispiel des LEGO-Systems (LegoRIS 2009).

LEGO Mindstorms Robotic Invention System (RIS)

Werkzeuge, die im Unterricht zum Einsatz kommen, sind das LEGO Mindstorms Robotic Invention System (RIS) und dessen Nachfolger LEGO Mindstorms NXT (siehe dazu Wiesner, Brinda 2007).

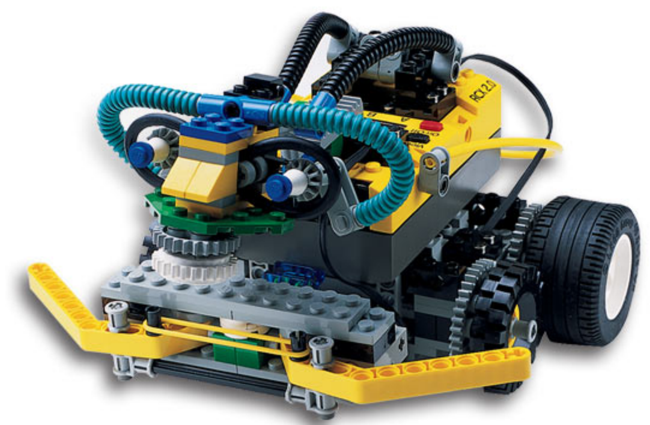


Abbildung 23: Lego Mindstorms RIS, Grafik aus http://shop.lego.com/Product/?p=3804, Zugriff am 10.7.2009

Das LEGO System hat Sensoren (mindestens einen Berührsensor und einen Lichtsensor, es gibt auch Geräuschsensoren und Sensoren zur Abstandsmessung). Als Aktoren dienen Motoren. Weiterhin sind Textausgaben und akustische Signale vor allem in der NXT-Serie möglich. Damit genügt das System den Anforderungen an ein technisches System in dieser Arbeit. Dieses technische System muss nun geeignet konfiguriert werden, um es für die Lösung einer bestimmten Aufgabe zu spezialisieren. Der Legoroboter kann z. B. darauf spezialisiert werden, einer schwarzen

Linie zu folgen oder rote und grüne Kugeln zu sortieren.
Die Programmiersprache ist graphisch und unterstützt einen kompositionellen Algorithmenentwurf, bei dem die Bausteine beliebig kombiniert werden können. Dieses Baukastenprinzip wird geeignet visualisiert. „LEGO Mindstorms RIS lässt sich visuell mittels RCX-Code programmieren. Die grafischen Symbole der Programmierumgebung repräsentieren stets vollständige algorithmische Grundelemente wie etwa Wiederholung oder bedingte Anweisung. Entsprechend der gestellten Aufgabe ordnen die Lernenden die notwendigen Symbole auf der Oberfläche an und stellen gegebenenfalls erforderliche Parameter in geeigneter Weise ein, beispielsweise Motordrehrichtungen. Damit können Syntaxfehler, wie sie aus der Skriptprogrammierung bekannt sind, etwa fehlende Zeichen, nicht entstehen. Die Programme sind grundsätzlich lauffähig und die Lernenden können sich auf die semantischen Probleme ihrer Konstruktion konzentrieren" (Wiesner, Brinda 2007).

Beispiel 28:

Abbildung 24: Beispielprogramm eines Schülers einer 7. Klasse

Solange der Lichtsensor an Eingang 2 „dunkel" registriert, wird bei Betätigung des Berührsensors an Eingang 1 die Richtung des Roboters umgekehrt. Wird der Berührsensor nicht getätigt, werden die Motoren an den Ausgängen A und C eingeschaltet.

Besonders sinnvoll ist hier die strikte Trennung zwischen Ein- und Ausgängen (Eingänge sind mit 1, 2, 3 beschriftet, Ausgänge mit A,B,C) und die Forderung, Motoren als Aktoren an A, B oder C zu schalten, während die Sensoren an die Eingänge geschaltet werden. Das System könnte verbessert werden, indem sich die Steckvorrichtungen für die Aktoren und Sensoren unterscheiden würden. (Ein Sensor kann dann gar nicht an einen Ausgang gesteckt werden.)
Da die Befehle dem imperativen Paradigma folgen und in Kapitel 2 angegeben worden ist, dass

imperative Programme orthogonalisiert sind, d. h. jede Kombination von Funktionen wieder eine gültige Konfiguration ist, genügt dieses Beschreibungsmittel den Ansprüchen einer Darstellungsform für Systemkonfigurationen. Leider ist der Umfang der Operationen im LEGO-RIS-System nicht so gering wie möglich, es gibt auch Mittel zur Abstraktion, nämlich Programme zu schreiben, sie zu benennen und in anderen Zusammenhängen wiederzuverwenden. Auch können Variablen universeller verwendet werden. Zur Konfiguration technischer Systeme sind nach den obigen Ausführungen nur boolesche Variablen notwendig. Dafür unterstützt die graphische LEGO-Programmierumgebung die Idee disjunkt paralleler Prozesse, wie Abbildung 25 zeigt.

Beispiel 29:

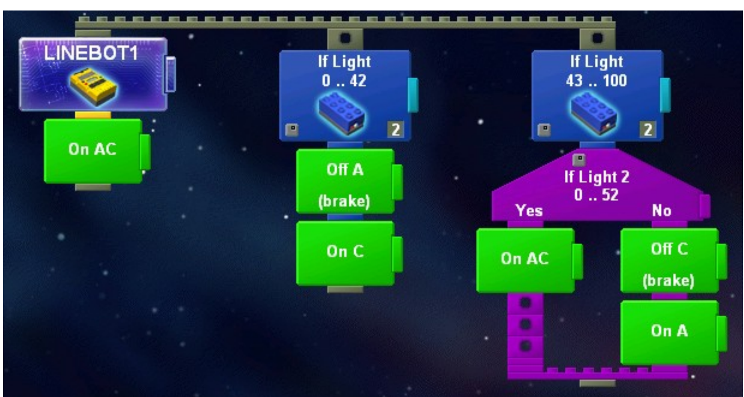


Abbildung 25: Nebenläufige Prozesse, Grafik aus http://www.baumfamily.org/dave/lego/books/ris2/guide.html, Zugriff 08.08.2009

Auch Wiesner und Brinda kommen zu dem Resultat: „Das Mindstorms-System erscheint speziell zur Einführung des Algorithmusbegriffs geeignet“ (Wiesner, Brinda 2007).

Vorteilhaft ist die einfache Übertragbarkeit auf Techniksysteme der Lebenswelt der Schülerinnen und Schüler. Einige technische Systeme, wie z. B. der Strichcodescanner können sogar mit dem Legosystem nachgebaut werden (siehe Beispiel 14).

Wurde weiter oben ein Übergang von graphischer funktionaler Programmierung zu textbasierter funktionaler Programmierung eher kritisch beleuchtet, zeigt Kapitel 6, wie der Übergang von graphischer imperativer Programmierung zu textbasierter imperativer Programmierung erfolgen kann.

Als Konfigurationssprachen technischer Systeme wurden in diesem Kapitel graphische funktionale Ansätze auf Basis endlicher Automaten und graphische imperative Ansätze als geeignet hervorgehoben. In Kapitel 2 sind die Schaltnetze und Schaltwerke als Konfigurationssprache an

einem Beispiel untersucht worden. Damit sind bereits drei Beschreibungsmittel für Systemkonfigurationen technischer Systeme gefunden:

- endliche Automaten bzw. Mealy-Maschinen
- Schaltnetze bzw. Schaltwerke
- graphische imperative Programmiersprachen

In Kapitel 3.6 wird noch ein weiteres graphisches Konstrukt im Mittelpunkt der Betrachtungen stehen. Dies soll aber nicht den Anschein erwecken, als ginge es darum, die Vor- und Nachteile gegeneinander aufzuwiegen und sich für einen Ansatz zu entscheiden.

Nach Meinung der Autorin ist es methodisch gesehen sehr sinnvoll, wenn mehrere Programmieransätze nebeneinander stehen bleiben. Kapitel 6 geht näher darauf ein. Die Schülerinnen und Schüler sind unterschiedliche Lerntypen, und jedes Paradigma begründet sich in einer anderen Sichtweise auf die Dinge. So mag es einerseits Schülerinnen und Schüler geben, denen der funktionale Ansatz angenehmer ist, und andererseits Lernende, die mit dem imperativen Ansatz besser zurechtkommen. Schule muss differenziertes Lernen zulassen (siehe u. a. Bönsch 2004) und dasselbe Lernmaterial auf jeweils unterschiedlichen Lernwegen präsentieren, damit jeder Lernende möglichst zu seinem Lerntyp passend gefördert werden kann.

Kapitel 6 zeigt außerdem, wie eine Erarbeitung der Algorithmenbausteine gelingen kann, wenn die Schülerinnen und Schüler vorher verschiedene Beschreibungsstrukturen kennen gelernt haben. Je unterschiedlicher die Beschreibungsmittel sind, desto besser lassen sich die Bausteine von Algorithmen als Gemeinsamkeiten erschließen. Weiterhin verknüpfen die Schülerinnen und Schüler viele verschiedene Gebiete der Informatik durch die Konfiguration in unterschiedlichsten Darstellungen (Schaltnetz, Mealy-Maschine, imperatives Programm, UML-Diagramm).

Die Arbeiten von Schubert und Schwill unterstützen dies: „Die Festlegung auf einen Baukasten (in der Regel ist das der imperative) reicht nicht aus. Hierüber herrscht allgemein Konsens“ (Schubert, Schwill 2004).

Haben sich die Schülerinnen und Schüler durch Konfigurationen in verschiedenen Sprachen die Algorithmenbausteine erschlossen, so sollten sie in der Lage sein, dieses Wissen auch zu transferieren. Die Lernenden sollten versuchen, Sensoren und Aktoren bekannter technischer Systeme ihrer Lebenswelt zu identifizieren und die Funktionalität algorithmisch zu beschreiben. Bekannte Mittel dazu sind Struktogramme.

3.6 Struktogramme vs. UML-Aktivitätsdiagramme

Wiesner und Brinda (Wiesner, Brinda 2007) wählen Struktogramme zur Programmdarstellung mit dem LEGO-System. Ein Ansatz, der von der Autorin im ersten Unterrichtsdurchgang auch gewählt wurde. Aber eignen sich Struktogramme wirklich, um ein Blutdruckmessgerät oder die Funktionsweise eines Wäschetrockners darzustellen? Ein entscheidender Nachteil wurde bereits mit der Diskussion der disjunkten Parallelität von Prozessen gegeben. Die Schülerinnen und Schüler können Nebenläufigkeit in Struktogrammen nicht unterbringen (siehe dazu Abbildung 16).
In diesem Zusammenhang sind die UML-Aktivitätsdiagramme zu nennen. Glichen sie in der UML 1.x noch den Flussdiagrammen, die schon immer eine Alternative zu den Struktogrammen darstellten, verfügen die UML-Aktivitätsdiagramme seit der Version 2.0 über „eine Petrinetz-ähnliche Semantik“ (Oesterreich 2006), und damit kann Nebenläufigkeit definiert werden.

Ein UML-Aktivitätsdiagramm besteht aus *Start-* und *Endknoten,* einer Reihe von *Aktions-* und *Kontrollknoten*, sowie *Kontrollflüssen,* mit denen diese Knoten verbunden sind.

Startknoten

Endknoten

Aktionsknoten, als Oval gekennzeichnet, stellen *elementare Ablaufschritte* dar. Im Zusammenhang mit technischen Systemen sind dies z. B. die Basisfunktionen.

Als Kontrollknoten sei die *Entscheidungsraute* genannt, die entsprechend der angegebenen Bedingung den Kontrollfluss verzweigt.

Bei einer *Synchronisation* wird auf alle eingehenden Kontrollflüsse gewartet, bevor der Kontrollfluss fortgesetzt wird.

Bei einer *Teilung* werden eingehende Kontrollflüsse ohne Bedingungen sofort in mehrere ausgehende nebenläufige Kontrollflüsse geteilt.

Das Erreichen eines Endknotens führt zum sofortigen Ende aller Aktionen im gesamten Diagramm, weshalb auch Regelkreise im Steuerungssystem unterbrochen werden, wie das folgende sehr vereinfachte Beispiel für einen Wäschetrockner zeigt:

Beispiel 30:

Der Regelkreis, der die Temperatur des Heizgebläses regelt, ist disjunkt parallel zum Steuerungssystem, welches so lange die Trommel dreht und das Gebläse angeschaltet lässt, wie die Wäsche feucht ist (siehe Abbildung 26).

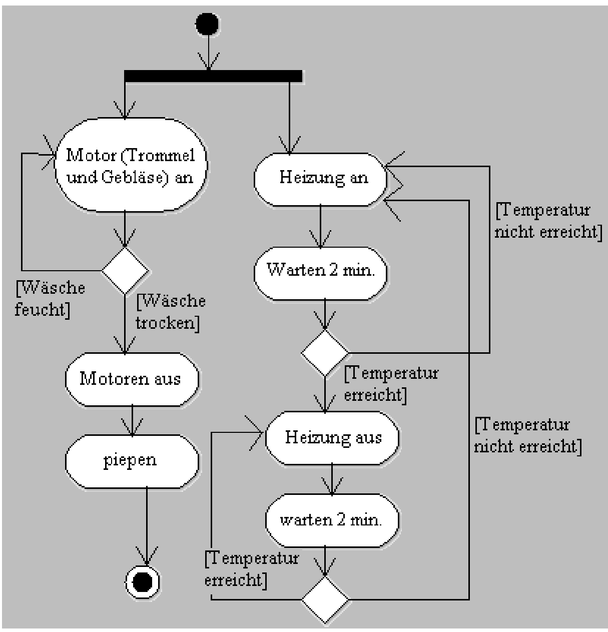


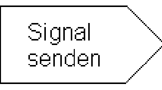
Abbildung 26: Funktionsweise eines Wäschetrockners

UML-Aktivitätsdiagramme bieten folgende Vorteile in der Syntax:

- Nebenläufigkeit wird integriert.
- Sie sind allgemein akzeptiert.
- Man kann den Schülerinnen und Schülern leicht folgende hilfreiche Struktur erklären: Immer wenn der Kontrollfluss mittels Rauten verzweigt wird, dann beziehen sich die Bedingungen an den Rauten entweder auf das Über- oder Unterschreiten bestimmter Sensorwerte, auf das Über- oder Unterschreiten einer Kombination von Sensorwerten oder auf (boolesche) Variablen. Die Aktionen des Systems stehen immer in Ovalen. Eingaben und Ausgaben werden so auch optisch unterschieden. In einem Oval kann niemals ein Sensorwert stehen. Während Funktionen, die die Aktoren ansteuern, niemals an Rauten stehen können.
- Optisch gesehen kann man in UML-Diagrammen Merkmale endlicher Automaten wiederfinden sowie Elemente imperativer Programme. Aus beiden Konzepten finden sich für den Lehrenden Übergangsmöglichkeiten.
- Durch die Pfeilrichtung wird die Überführung der Eingabewerte in das System zu den

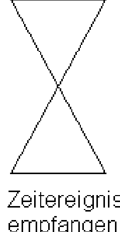
Ausgabewerten optisch unterstützt.

Zwei zu konfigurierende Systeme wurden mit Automaten-Kara und dem LEGO-RIS-System vorgestellt. Dabei ist das LEGO-System ein reales System, bei dem der Computer nur zur Angabe des Algorithmus dient. Bei Automaten-Kara dient er auch zur Simulation des Systems. In Kapitel 5 wird die Lernumgebung PuMa vorgestellt, die wie das LEGO-System ein reales System darstellt. Mit geeigneten Compilern kann man den Algorithmus der Konfiguration in solch realen Systemen auch als UML-Aktivitätsdiagramm angeben, und Software übersetzt dies in die entsprechenden Befehle. Kapitel 5 und 6 gehen näher darauf ein. In diesem Fall sollten noch weitere Elemente des UML-Aktivitätsdiagramms Verwendung finden, die geeignet visualisieren, dass Eingaben von außen kommen und Ausgaben nach außen gesendet werden (nach Oesterreich 2006):



„Das Senden eines Signals wird dargestellt durch ein Rechteck, das auf einer Seite spitz ist.“

„Bei Signalen die empfangen werden, wird ein Rechteck gezeichnet, bei dem auf einer Seite eine Spitze hineinragt.“



„Für Zeitereignisse kann alternativ ein Sanduhrsymbol verwendet werden.“

Abbildung 27 zeigt die Konfiguration eines Systems mit Hilfe der Syntax der UML-Aktivitätsdiagramme. Das System besteht aus einem Taster, der an Eingang 1 angeschlossen ist, und einer Lampe, die an Ausgang 2 angeschlossen ist. Die Konfiguration soll das System so spezialisieren, dass der Taster sowohl zum Ein- wie auch zum Ausschalten der Lampe verwendet werden kann. Die Pfeilrichtungen der Signalausgaben und -eingaben visualisieren die Flussrichtung im System.

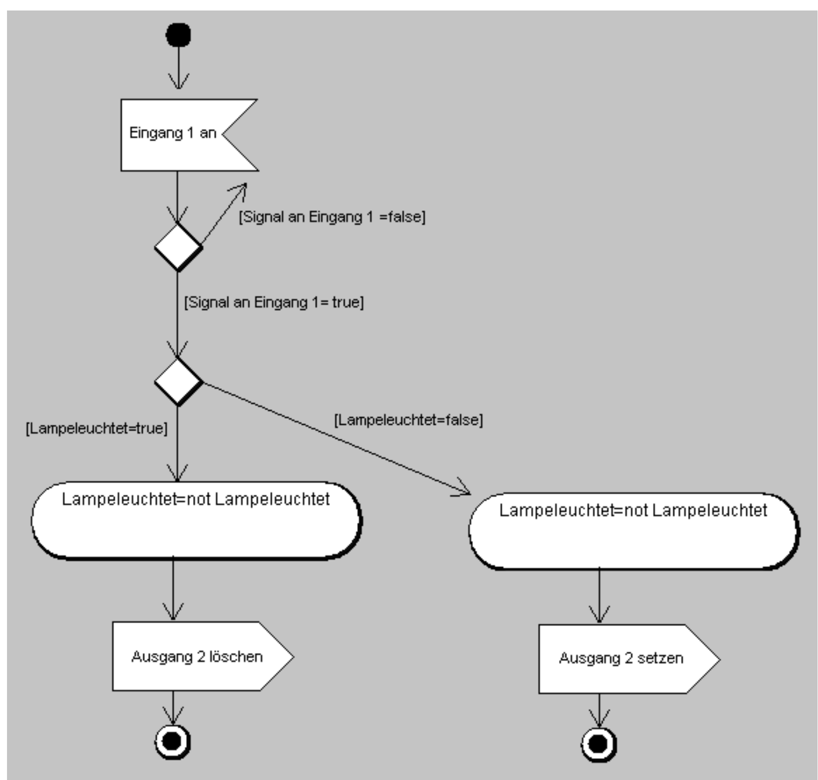


Abbildung 27: Beispiel eines UML-Aktivitätsdiagramms

3.7 Zusammenfassung

Mit „Programmieren für Alle“ besteht die Forderung, allgemeinbildende Inhalte im Unterricht zu vermitteln, die den Schülerinnen und Schülern helfen, die sie umgebende Welt besser zu durchdringen und einordnen zu können. Da Informatik an Schulen eins der wenigen Fächer ist, das mit technischem Bezug unterrichtet werden kann, liegt es nahe, den Allgemeinbildungsanspruch zu wahren, indem die Schülerinnen und Schüler befähigt werden, die sie umgebende *technische* Welt besser zu verstehen, z. B. die Funktionsweise technischer Systeme ihrer Umwelt algorithmisch beschreiben zu können. Technische Systeme besitzen Sensoren und Aktoren. Für die Systemkonfiguration und ihre Beschreibungsmittel ergeben sich dabei einige Besonderheiten, die in diesem Kapitel herausgearbeitet wurden:

- Graphische Programmierumgebungen sind für die Konfiguration technischer Systeme ausreichend, was den Unterricht entlastet und einem Unterricht „für Alle“ gerechter wird.
- Technische Systeme lassen sich mit Schaltwerken oder Zustandsgraphen beschreiben. Dadurch können die nötigen Basisoperationen weiter eingeschränkt werden. Bei imperativen Programmen sind z. B. boolesche Variablen ausreichend.

- Konfigurationen technischer Systeme erfordern Beschreibungsmittel, die eine disjunkte Parallelität von Funktionen unterstützen.
- Technische Systeme lassen sich auch in der Realität mit Hilfe von Schaltwerken bzw. Zustandsgraphen konfigurieren. Das schränkt die Komplexität der Aufgaben ein und definiert, wie in Kapitel 5 weiter ausgeführt werden wird, nach Meinung der Autorin einen Rahmen für Aufgabenklassen in einer Unterrichtseinheit „Programmieren für Alle“ der Sekundarstufe I.
- Konfigurationen technischer Systeme lassen sich außer mit Schaltwerken/Schaltnetzen und Zustandsgraphen (Mealy-Maschinen) auch mit Hilfe von imperativen Programmen und UML-Aktivitätsdiagrammen angeben.

4 Zur Methodik einer Unterrichtseinheit „Programmieren für Alle“

In diesem Kapitel soll die Methodik des Programmierunterrichts näher beleuchtet werden. Dabei wird der Schwerpunkt auf zwei wesentliche Aspekte gelegt:

- Methoden, die sich auf das Erarbeiten einer der Aufgabenstellung entsprechenden Konfiguration beziehen
- grundsätzliche methodische Entscheidungen über die Unterrichtskultur eines Unterrichts im „Programmieren“ in der Sekundarstufe I

4.1 Methodik zum Erarbeiten der Systemkonfigurationen

In dieser Arbeit soll aufgezeigt werden, wie wichtig es für die Schülerinnen und Schüler ist, die der Aufgabenstellung entsprechenden Konfigurationen technischer Systeme *eigenständig* zu finden bzw. zu erarbeiten. Die Arbeit grenzt sich damit gegen einen aktuellen Ansatz in der Informatikdidaktik, die „Dekonstruktion von Systemen“ (siehe Hampel, Magenheim, Schulte 1999) ab. Begründet werden soll der Ansatz der eigenständigen Algorithmensuche u. a. mit Hilfe aktueller Forschungsergebnisse der Mathematikdidaktik.

4.1.1 Dekonstruktion von Systemen

Hampel, Magenheim und Schulte begründeten die Vorgehensweise der „Dekonstruktion von Systemen“ für die Sekundarstufe II. Sie schreiben: „Anstelle der Neuimplementation der eigenen Modellierung dekonstruieren die Lernenden eine Implementation, die auf den Analyse-, Design- und Implementationsentscheidungen eines anderen „Entwicklungsteams“ beruht“ (zitiert nach Brinda 2004). „Anhand einer in Java codierten Software, mit hinreichender, jedoch unter didaktischen Gesichtspunkten reduzierter Komplexität, werden Systemfunktionen erkundet und das der Software implizite Modell eines Realitätsausschnitts so weit wie möglich expliziert und mit einem realen System verglichen“ (Hampel, Magenheim, Schulte 1999).
Vereinfachend werden also Systemkonfigurationen, die bereits implementiert sind, von den Schülerinnen und Schülern nachvollzogen. Systemkonfigurationen werden bei dieser methodischen Vorgehensweise nicht selbstständig erarbeitet, sondern die Lernenden üben sich im Verstehen fertiger Programme, um ihr Verständnis des zugrunde liegenden Systems zu verbessern.
Diese Methodik hat seine Berechtigung, wenn man argumentiert, dass reale Systeme zu komplex sind, um von Schülerinnen und Schülern erfolgreich rekonstruiert zu werden, die Beschäftigung mit realen Systemen aber ein vertiefendes Verständnis für die Lebenswelt der Schülerinnen und Schüler fördert. Magenheim, Hampel und Schulte sehen den Nachteil des „Selbermachens“ in der geringen

Komplexitätstiefe der zu lösenden Probleme und befürchten dadurch eine Entfernung von der Realität.

Diese Arbeit verfolgt einen anderen Ansatz. Bei der „Dekonstruktion von Systemen“ wird vorhandene Software von den Lernenden nachvollzogen, weil nur so Systeme der Realität Unterrichtsgegenstand sein können. Der Grundgedanke ist, dass Systeme der Lebenswelt zu komplexen Aufgabenstellungen führen, die Schülerinnen und Schüler nicht bewältigen können. In dieser Arbeit beschränken wir uns aber auf technische Systeme der Lebenswelt. Nach den Ausführungen in Kapitel 3 sind Konfigurationen bekannter *technischer* Systeme des Alltags der Lernenden in der Komplexität stark beschränkt, nämlich auf Mealy-Maschinen oder einfache Schaltnetze. Kapitel 6 zeigt in Unterrichtsergebnissen, dass Schülerinnen und Schüler die Komplexität dieser Aufgaben auch bewältigen können, wenn sie die notwendigen Algorithmen selbstständig entwickeln. Als weiteres Beispiel sei die von Schülerinnen und Schülern eigenständig entwickelte Konfiguration eines Strichcodescanners aus Beispiel 14 genannt. Voraussetzung einer eigenständigen Algorithmensuche sind die in Kapitel 2 beschränkten Beschreibungsmittel der Systemkonfiguration, die aus einer minimalen Anzahl an Basiselementen und Operationen bestehen. Die nächsten Abschnitte gehen näher auf diesen Aspekt ein. Folgender Unterschied in der Herangehensweise ist aber dieser:

Bei der „Dekonstruktion von Systemen“ ist das Produkt vorhanden, bei der Systemkonfiguration erzeugen die Schülerinnen und Schüler ein Produkt oder rekonstruieren bekannte Elemente.

An dieser Stelle stellt sich also die Frage: „Selber machen oder nur verstehen?“

4.1.2 Erkenntnisse aus der Mathematikdidaktik

Die Mathematikdidaktikerin Regina Bruder schreibt: „Es werden im Folgenden acht Aufgabentypen vorgestellt, die sich aus unterschiedlicher Bekanntheit von Anfangs- und Zielsituation und möglichen Lösungswegen ergeben – das sind die drei Komponenten, mit denen jede Aufgabe in ihrer Struktur beschrieben werden kann. Unter den *Komponenten* einer Aufgabe sollen hier verstanden werden:

1. die *Anfangssituation*: Voraussetzungen, gegebene Größen, Informationen zu einem Sachverhalt o. Ä.
2. *Transformationen*, die die Anfangssituation in die Endsituation überführen bzw. die von dem Gegebenen zum Gesuchten hinführen: Lösungsweg(e), mathematische Modelle, Beweiskette …
3. Die *Endsituation*: Gesuchtes, Behauptung, Schlussfolgerungen, Resultate usw.“

(Bruder, Leuders, Büchter 2008).

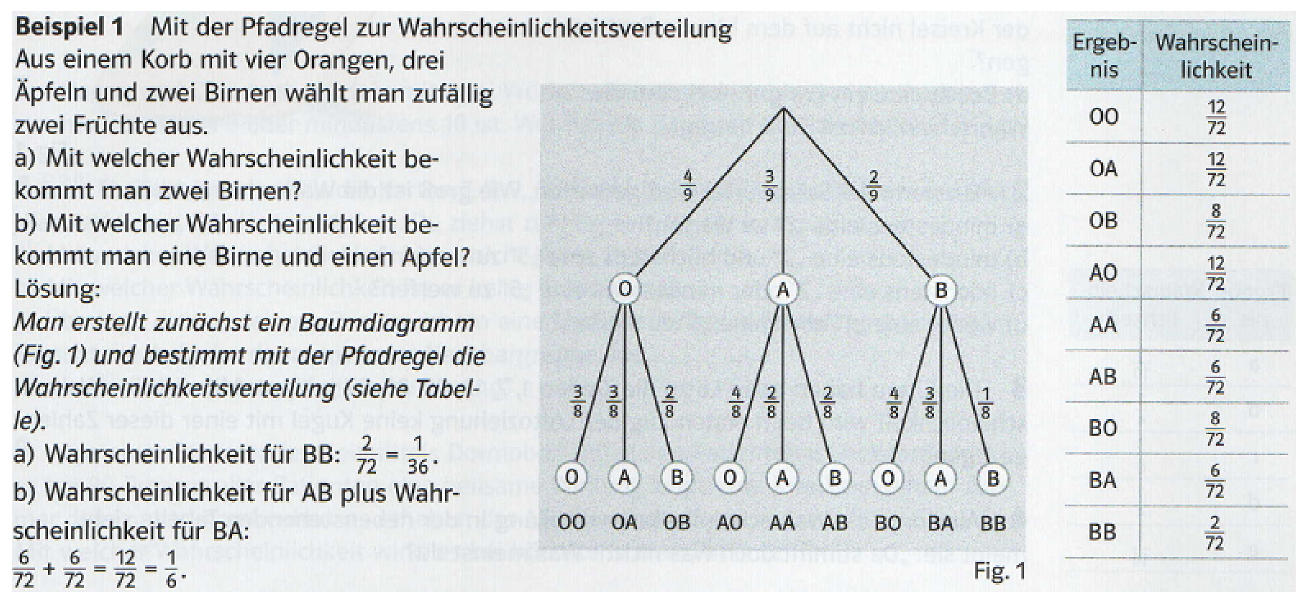
Die in Abbildung 28 angegebenen acht Aufgabentypen entstehen, „wenn man alle Möglichkeiten durchspielt, ob die drei Komponenten einer Aufgabe jeweils bekannt/vorgegeben/verfügbar sind (X) oder nicht (-)“ (Bruder, Leuders, Büchter 2008).

Gegebenes	Transformationen	Gesuchtes	Bezeichnung des Aufgabentyps	Beispielaufgabe
×	×	×	gelöste Aufgabe, Musteraufgabe, Aufgabe zur Fehlersuche	– Stimmt das?… – Wo steckt der Fehler?
×	×	–	einfache Bestimmungsaufgabe **(Grundaufgabe)**	– Löse die quadratische Gleichung $3x^2 - 7x = 8$. – Kopfrechenaufgaben – Berechne das Volumen einer Halbkugel mit dem Radius von 5 cm.
–	×	×	einfache **Umkehraufgabe**	– Gib eine quadratische Gleichung an, die 2 und −3 als Lösungen hat. – Bestimme den Radius einer Kugel, die 30 cm³ Volumen hat. – Zahlenrätsel: Mit einer gedachten Zahl werden bestimmte bekannte Rechenoperationen ausgeführt und das Ergebnis wird genannt. Die gedachte Zahl soll bestimmt werden.
×	–	×	Beweisaufgabe, Spielstrategie finden	– Beim Nimm-Spiel gewinnt Frank immer. Wie macht er das? *Es liegen 20 Streichhölzer auf dem Tisch. Zwei Spieler spielen gegeneinander. Gewonnen hat derjenige, der das letzte Streichholz nehmen kann, wenn entweder ein, zwei oder drei Hölzer pro Zug genommen werden dürfen.* – Warum ist die *p-q*-Formel zur Lösung quadratischer Gleichungen immer richtig?
×	–	–	schwere Bestimmungsaufgabe, auch: Teil einer gestuften Aufgabe (Blütenmodell)	Ist eine Tetra-Pak-Milchtüte verpackungsoptimal gestaltet?
–	–	×	schwierige Umkehraufgabe, Modellierungsproblem mit Zielvorgabe	Ein Teich soll eine Fläche von ca. 10 m² erhalten.
–	×	–	Aufforderung, eine Aufgabe zu einem gegebenen mathematischen Werkzeug zu erfinden	Erfinde Beispielaufgaben zu den drei typischen Fragestellungen der Prozentrechnung.
–	–	–	Problemsituation mit offenem Ausgang (Trichtermodell)	Führe eine Befragung zu einem gegebenen Thema bei deinen Mitschülern durch und stelle die Ergebnisse vor.

Abbildung 28: Acht zentrale Aufgabentypen für nachhaltiges Lernen mit Beispielen (aus Bruder, Leuders, Büchter 2008)

In einem typischen Mathematik-Schulbuchwerk der Sekundarstufe I (Lambacher Schweizer 2006) findet man in jedem Kapitel ein gleiches Vorgehensschema. Einleitend wird an einem Problem ein neuer Lösungsweg für einen neuen Aufgabentyp vorgestellt. An ein bis zwei Beispielen wird dieser neue Lösungsweg dann zunächst beispielhaft an einer Aufgabe vorgeführt.

Beispiel 31:



Beispiel 1 Mit der Pfadregel zur Wahrscheinlichkeitsverteilung
Aus einem Korb mit vier Orangen, drei Äpfeln und zwei Birnen wählt man zufällig zwei Früchte aus.
a) Mit welcher Wahrscheinlichkeit bekommt man zwei Birnen?
b) Mit welcher Wahrscheinlichkeit bekommt man eine Birne und einen Apfel?
Lösung:
Man erstellt zunächst ein Baumdiagramm (Fig. 1) und bestimmt mit der Pfadregel die Wahrscheinlichkeitsverteilung (siehe Tabelle).
a) Wahrscheinlichkeit für BB: $\frac{2}{72} = \frac{1}{36}$.
b) Wahrscheinlichkeit für AB plus Wahrscheinlichkeit für BA:
$\frac{6}{72} + \frac{6}{72} = \frac{12}{72} = \frac{1}{6}$.

Fig. 1

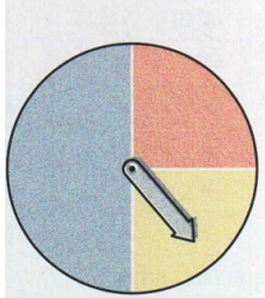
Ergebnis	Wahrscheinlichkeit
OO	$\frac{12}{72}$
OA	$\frac{12}{72}$
OB	$\frac{8}{72}$
AO	$\frac{12}{72}$
AA	$\frac{6}{72}$
AB	$\frac{6}{72}$
BO	$\frac{8}{72}$
BA	$\frac{6}{72}$
BB	$\frac{2}{72}$

Abbildung 29: gelöste Beispielaufgabe (aus Lambacher Schweizer 2006)

In Beispiel 31 finden wir eine Aufgabe, in der sowohl die Anfangssituation als auch die Transformation und Endsituation beschrieben sind. Nach Bruder (Bruder, Leuders, Büchter 2008) handelt es sich hier um eine „gelöste Aufgabe“ oder „Musteraufgabe“.

Anschließend folgt in diesem Buch (Lambacher Schweizer 2006) ein Block von Aufgaben, in dem dieser neue Lösungsweg geübt werden soll, siehe Beispiel 32. Charakteristisch ist, dass jeweils der Startzustand gegeben ist. Da ja der *neue* Lösungsweg dieser Einheit geübt werden soll, ist die Transformation damit ebenfalls vorgegeben. Die Schülerinnen und Schüler sollen jetzt mit dieser Transformation und der gegebenen Anfangssituation das Ergebnis, das Ziel, berechnen. Nach Bruder (Bruder, Leuders, Büchter 2008) handelt es sich dabei um eine „einfache Bestimmungsaufgabe“.

Beispiel 32:



5 Das Glücksrad in Fig. 2 wird zweimal gedreht.
a) Gib die Wahrscheinlichkeitsverteilung an.
b) Wie groß ist die Wahrscheinlichkeit dafür, dass zweimal rot angezeigt wird?
c) Mit welcher Wahrscheinlichkeit wird zweimal die gleiche Farbe angezeigt?
d) Mit welcher Wahrscheinlichkeit wird mindestens einmal gelb angezeigt?
e) Mit welcher Wahrscheinlichkeit wird beim zweiten Drehen blau angezeigt?
f) Vervollständige: Mit 75 % Wahrscheinlichkeit wird …

Abbildung 30: einfache Bestimmungsaufgabe (aus Lambacher Schweizer 2006)

Die Aufgaben sind natürlich unterschiedlich komplex, teilweise anwendungsorientiert, aber durch die Gliederung der Kapitel immer dem neu zu erlernenden Aufgabentyp des jeweiligen Kapitels zugeordnet. Für einen Großteil der Aufgaben einiger Schulbücher im Mathematikunterricht der Sekundarstufe I gilt also, dass den Schülerinnen und Schülern bei der Suche nach einem Ergebnis die anzuwendende Transformation bekannt ist.

Verfolgt man die aktuelle Diskussion der Mathematikdidaktik, dann besteht eine Forderung nach mehr *offenen Aufgaben* im Unterricht. Offene Aufgaben sind dabei so zu verstehen, dass Schülerinnen und Schüler vor ein Problem gestellt werden, das auf unterschiedliche Weise gelöst werden kann und bei dem es oft mehr als eine „richtige“ Lösung gibt. Die Transformation, die Anfangssituation und auch das Ergebnis sind nicht gegeben und müssen von den Lernenden gefunden werden.
Beispiel 33 zeigt eine sogenannte offene Aufgabe (aus Herget, Jahnke, Kroll 2008).
Wilfried Herget (Herget, Jahnke, Kroll 2008) schreibt dazu: „Das Zeitungsfoto mit dem Riesen-schuh dient als Auslöser für die Frage: „Welche Schuhgröße hat dieser Riesenschuh?“ Eine solche Aufgabe ist ungewohnt und es ist immer sehr spannend, welche verschiedenen Lösungswege dazu gefunden werden.“
Solche Aufgabenstellungen und Vorgehensweisen sind aber bislang nur sehr vereinzelt bis gar nicht in den aktuellen Schulbüchern zu finden.

Beispiel 33:

Auf großem Fuß müsste leben, wem dieser Riesenschuh passt. Antal Annus, ein 73 Jahre alter Schuhmacher aus dem ungarischen Dorf Csanádapáca, zeigt stolz sein beeindruckendes Werk. Ob er den Schuh jedoch für einen seiner Kunden maßgeschneidert hat, ist nicht bekannt.

Abbildung 31: Beispiel einer offenen Aufgabe (aus Herget, Jahnke, Kroll 2008)

Der aktuelle Mathematikunterricht ist somit davon gekennzeichnet, dass größtenteils Aufgaben mit bekannter Transformation gestellt werden. Diese Aussage wird auch durch die Arbeiten von Johanna Neubrand unterstützt. Sie klassifiziert (Neubrand 2002) ebenfalls die Aufgabentypen im Mathematikunterricht:

	Art der Aufgabe	Ausgangs-zustand	Bearbeitung, Lösungsweg	Zielzustand
Aufgabenart 1 (verallgemeinerte Bestimmungsauf-gaben)	Bestimmungsaufgabe	vorgegeben	gesucht	gesucht
	Umkehraufgabe	gesucht	gesucht	vorgegeben
Aufgabenart 2 (verallgemeinerte Grundaufgaben)	Grundaufgabe	vorgegeben	vorgegeben	gesucht
	Umkehraufgabe	gesucht	vorgegeben	vorgegeben
Aufgabenart 3 (Beweisaufgaben)	Beweisaufgabe	vorgegeben	gesucht	vorgegeben
Aufgabenart 4 (Reflexionsaufga-ben)	Aufgabe über Aufgaben	vorgegeben	vorgegeben	vorgegeben
	Selbst eine Aufgabe bilden	gesucht	vorgegeben	gesucht

Abbildung 32: Unterschiedliche Aufgabenarten im Mathematikunterricht (aus Neubrand 2002)

Unter Aufgabenart 3 (Beweisaufgaben) finden wir in Abbildung 32 die Analogie zur Aufgabe bei Bruder (Bruder, Leuders, Büchter 2008), wo Anfangs- und Endzustand gegeben sind und eine Transformation gefunden werden muss, die den Anfangszustand in den Endzustand transformiert. Johanna Neubrand hat Unterrichtssituationen in Deutschland, den USA und Japan untersucht und kommt zu folgendem Resultat:

N = 833	USA	Deutschland	Japan
Arithmetik			
verallgemeinerte Bestimmungsaufgaben	34 (42 %)	19 (66 %)	
verallgemeinerte Grundaufgaben	46 (57 %)	10 (34 %)	
Beweisaufgaben			
Reflexionsaufgaben	1 (1 %)		
gesamt (Arithmetik)	81 (100 %)	29 (100 %)	
Algebra			
verallgemeinerte Bestimmungsaufgaben	147 (80 %)	115 (93 %)	33 (69 %)
verallgemeinerte Grundaufgaben	36 (20 %)	6 (5 %)	13 (27 %)
Beweisaufgaben		1 (1 %)	
Reflexionsaufgaben		1 (1 %)	2 (4 %)
gesamt (Algebra)	183 (100 %)	123 (100 %)	48 (100 %)
Geometrie			
verallgemeinerte Bestimmungsaufgaben	146 (74 %)	80 (70 %)	25 (43 %)
verallgemeinerte Grundaufgaben	48 (24 %)	30 (26 %)	2 (3 %)
Beweisaufgaben		1 (1 %)	23 (40 %)
Reflexionsaufgaben	2 (1 %)	4 (4 %)	8 (14 %)
gesamt (Geometrie)	196 (100 %)	115 (100 %)	58 (100 %)
gesamt	460	267	106

Abbildung 33: Verteilung der verschiedenen Aufgabenarten im Unterricht in Deutschland, den USA und Japan (aus Neubrand 2002)

In Deutschland finden wir demnach überwiegend Aufgabenstellungen, die im Bruder´schen Schema denen entsprechen, deren Transformation bekannt ist, wie Abbildung 33 zeigt. Außerdem sind nur ungefähr 1% der Aufgaben im Mathematikunterricht in Deutschland Beweisaufgaben, also solche, bei der die Transformation gesucht werden muss.

Das Finden einer Systemkonfiguration im Programmierunterricht „für Alle“ ist in Kapitel 2 als das Finden einer Funktion oder Transformation definiert worden, die Eingabewerte auf Ausgabewerte abbildet. Um die Funktionsweise technischer Systeme zu modulieren, muss nach den Überlegungen aus Kapitel 2 ein Endzustand des Systems detailliert spezifiziert und gegeben sein. Meist sind in den Aufgaben (siehe Beispiel 26) auch Anfangssituationen gegeben. Nach dem Bruder´schen Schema ergibt das die folgende Konstellation:

Gegebenes	Transformation	Gesuchtes	Bezeichnung des Aufgabentyps
X	-	X	Beweisaufgabe, Spielstrategie finden

Tabelle 4: Einordnung der Aufgabe, eine Systemkonfiguration zu finden, gemäß Bruder

Also einen Aufgabentyp, der als Beweisaufgabe nach Neubrand (Neubrand 2002) eingestuft wird und in Deutschland nach Abbildung 33 im Mathematikunterricht bislang *kaum* zu finden ist.
Das eigenständige Suchen einer Systemkonfiguration erweitert das Spektrum der den Schülerinnen und Schülern gestellten Aufgabentypen und eröffnet ihnen ein Übungsfeld, das z. Zt. durch den Mathematikunterricht nur unzureichend abgedeckt ist.
Damit zeigt diese Arbeit, dass gerade die *eigenständige* Algorithmensuche die Anforderungen an die kognitiven Leistungen der Schülerinnen und Schüler tatsächlich um einen neuen wesentlichen Aspekt erweitert, der im bisherigen Unterricht anderer Fächer wenig Beachtung findet. Ein Unterricht im Programmieren „für Alle" ist dann besonders sinnvoll, wenn Kompetenzen vermittelt werden, die in anderen Schulfächern oder Unterrichtseinheiten in dieser Form kaum zu finden sind.

Die Strategie der „Dekonstruktion von Systemen" kann in das Bruder´sche Schema auch eingeordnet werden. Beim Nachvollziehen fertiger Programme sind Anfangssituation, Transformation und Endsituation gegeben. Nach dem Bruder´schen Schema ergibt das eine Konstellation wie folgt:

Gegebenes	Transformation	Gesuchtes	Bezeichnung des Aufgabentyps
X	X	X	Gelöste Aufgabe, Musteraufgabe

Tabelle 5: Einordnung der Strategie „Dekonstruktion von Systemen" nach Bruder

Die Klasse von Aufgaben gehört nach Bruder zu den „einfachen Aufgaben" und damit Aufgaben, die *gehäuft* im bisherigen Mathematikunterricht in Deutschland zu finden sind, siehe (Neubrand 2002). Damit ist der Ansatz der „Dekonstruktion von Systemen" sinnvoll, weil er einen Unterricht „für Alle" ermöglicht. Der Anspruch einer eigenständigen Algorithmensuche scheint sehr hoch zu sein, da die Unterrichtserfahrung zeigt, dass viele Schülerinnen und Schüler hier scheitern, obgleich sie in anderen Themengebieten der Informatik erfolgreich mitarbeiten. Nach einigen Beschränkungen der Kapitel 2 und 3, die im folgenden nochmals aufgeführt werden, ist der Weg der eigenständigen Algorithmensuche aber gangbar, wie Unterrichtserfahrungen zeigen.
Zusammenfassend lässt sich Folgendes resümieren: Nach Meinung der Autorin müssen Schülerinnen und Schüler die der Aufgabenstellung entsprechenden Systemkonfigurationen

technischer Systeme *eigenständig* entwickeln. Es ist eine Anforderung an die Art der Aufgaben, das möglich zu machen. Darauf wird in Kapitel 5 wieder Bezug genommen, wenn die Art der Aufgaben bei Systemkonfigurationen klassifiziert wird. Zwei didaktische Entscheidungen früherer Kapitel ermöglichen die Methode der selbstständigen Algorithmensuche:

1. Ein Ergebnis aus Kapitel 2 ist die Forderung, bei der Beschreibung von Systemkonfigurationen Funktionen beliebig kombinieren zu können, wobei jede Kombination zu einem lauffähigen, wenn auch „falschen" Ergebnis führt. Da die Anzahl der Funktionen begrenzt ist, können insbesondere schwächere Schülerinnen und Schüler, die Schwierigkeiten mit der gewünschten analytischen Problemlösung haben, auch mit einfachem Experimentieren zu Ergebnissen kommen.
2. Ein Ergebnis aus Kapitel 3 ist die Forderung nach graphischen Programmierumgebungen, in denen keine Syntaxfehler produziert werden können. Damit wird eine eigenständige Algorithmensuche vereinfacht, weil die Schülerinnen und Schüler nicht erst überlegen müssen, ob es sich um syntaktische oder logische Fehler handelt.

Für eine erfolgreiche eigenständige Konfigurationssuche ist der Umgang der Schülerinnen und Schüler mit Fehlern von entscheidender Bedeutung. Deshalb widmet sich der nächste Abschnitt diesem Aspekt.

4.1.3 Umgang mit Fehlern

Die Verwendung graphischer Beschreibungssprachen schließt syntaktische Fehler aus. Wenn die Schülerinnen und Schüler eigenständig eine Idee entwickelt, eine Transformation gefunden haben, die ihrer Meinung nach das Problem löst, und das Programm funktioniert nicht in gewünschter Weise, dann müssen die Lernenden sich eingestehen, nicht präzise genug gearbeitet zu haben. Sie müssen lernen, ihre Meinung zu revidieren und weitere Alternativen bedenken. Die Schülerinnen und Schüler müssen also, um mit Heymanns Worten zu reden, ihre Vorstellungen und Ideen klar formulieren und sich Schwächen und Fehler eingestehen (Heymann 1996 in Bezug auf die „Stärkung des Schüler-Ichs").

Um gestärkt aus dieser Situation hervorzugehen und schließlich doch zu einem positiven Resultat zu kommen, darf nach Meinung der Autorin nicht der Lehrende als „Allwissender" mit dem Finger auf den Fehler zeigen, sondern den Schülerinnen und Schülern muss die Möglichkeit zu einer eigenständigen, zielgerichteten Fehlersuche gegeben werden. Daraus ergibt sich eine *weitere* Forderung an die Programmierumgebung. Unterstützt wird diese Forderung von der Aussage einer Schülerin der 11. Klasse, die in einer freien schriftlichen Evaluation angibt:

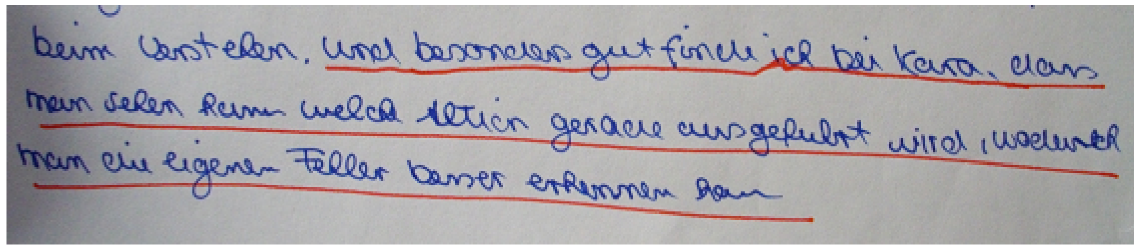


Abbildung 34: Aussage einer Schülerin zur Fehlersuche

„...und besonders gut finde ich bei Kara, dass man sehen kann, welche Aktion gerade ausgeführt wird, wodurch man die eigenen Fehler besser erkennen kann.“

Automaten-Kara (Reichert, Nievergelt, Hartmann 2005), wie in Kapitel 3.5.2.1 vorgestellt, zeigt während der Programmausführung an, in welcher Zeile oder bei welcher Anweisung es sich gerade in der Ausführung befindet. Weiterhin läuft das Programm auch fehlerbehaftet. Erst wenn der Marienkäfer aufgrund eines Programmierfehlers nicht „weiß, was zu tun ist“, bricht das Programm ab und in einem Dialogfenster wird der genaue Fehler beschrieben.

Solche Konzepte sollten sinnvoll in eine Entwicklungsumgebung integriert sein. Bei der Erstellung graphischer Programmierumgebungen sollte der aktuell ausgeführte Befehl angezeigt oder markiert werden.

4.2 Methodische Überlegungen bzgl. der Unterrichtseinheit „Programmieren“

Inhaltlich soll nach dem Ansatz dieser Arbeit in einem Unterricht „Programmieren für Alle“ die Funktionsweise technischer Systeme der Lebenswelt algorithmisch nach dem Ansatz der Systemkonfiguration beschrieben werden. Als methodische Grundkonzepte eines solchen Unterrichts sind nach Meinung der Autorin drei Punkte hervorzuheben, auf die im Folgenden eingegangen wird:

- zweckorientierter Unterricht, der konkrete Produkte erzeugt
- fächerübergreifender Unterricht
- Projektunterricht

4.2.1 Zweckorientierter Unterricht mit konkreten eigenen Produkten

Systemkonfigurationen sind per Definition zweckorientiert. In Kapitel 2 wurde gezeigt, dass eine detaillierte Spezifikation als Endzustand oder Aufgabe vorliegen muss und das System, in dem diese Aufgabe platziert ist, intuitiv erfassbar sein muss. Die Schülerinnen und Schüler konfigurieren das System für den angegebenen Zweck und nutzen dabei die in Kapitel 3 angegebenen

Beschreibungsmittel. Am Ende steht ein konkretes Produkt.

Beispiel 34:

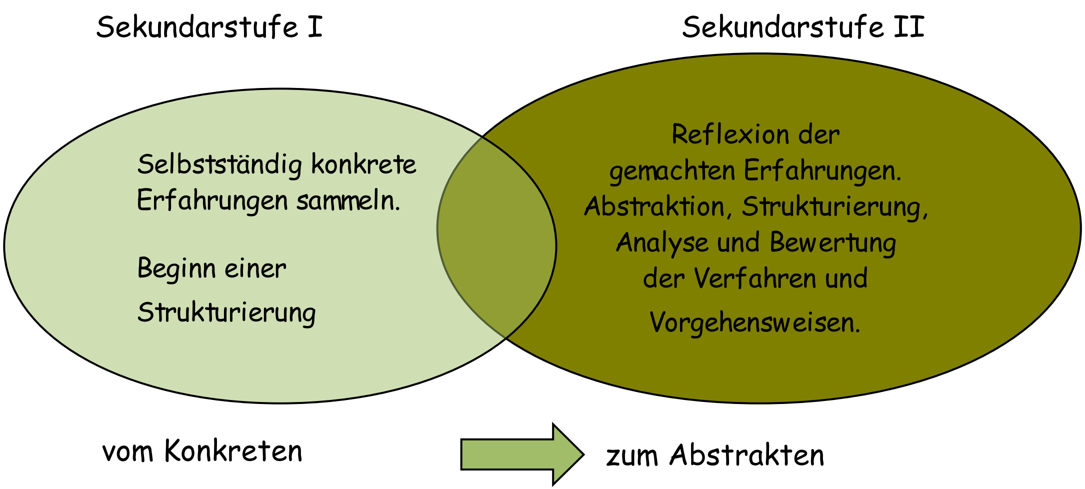
Die Autorin konnte folgende Beobachtung in einer Grundschule machen, die bereits ab der ersten Klasse den Computer einsetzt: Die Schülerinnen und Schüler arbeiten dort mit einer Software, die sich „Lernwerkstatt" nennt. Dort kann man zwischen mehreren Programmen auswählen. Wählt der Lernende z. B. das Programm „Rechnen", dann erscheinen Rechenaufgaben, die der Lernende lösen muss. Die Schülerinnen und Schüler erhalten sofort eine positive oder negative Rückmeldung. Überspitzt kann man diese Lernmethode folgendermaßen charakterisieren: „Der Computer weiß alles. Die Schülerinnen und Schüler nicht."

Im Gegensatz dazu verwenden Schülerinnen und Schüler zum Programmieren oder zur Konfiguration technischer Systeme den Computer als Werkzeug. Der Rechner führt dabei die Befehle der Lernenden aus und dient als Befehlsempfänger. In diesem Kontext „wissen die Schülerinnen und Schüler alles, nämlich wie das System zu konfigurieren ist, und der Computer weiß nichts".

Die Schülerinnen und Schüler müssen den Computer als Werkzeug nutzen, ihre eigenen Ideen umzusetzen. Aber nicht nur dieser Aspekt „entthront" den Rechner. Das System steht im Mittelpunkt des Interesses, und hier sollten Systeme gewählt werden, die nicht nur virtuell, sondern *real* sind. Ein Beispiel für solch ein System ist die LEGO-Mindstorms-Umgebung, die in Kapitel 3.5.2.2 vorgestellt wurde. Konfigurieren die Schülerinnen und Schüler das LEGO-System, dann tut sich etwas, es bewegt sich etwas, es werden immer funktionierende lauffähige Produkte erzeugt, was Produktstolz unabhängig vom Urteil des Lehrenden hervorruft. Ein weiteres Beispiel ist das PuMa-System, das in Kapitel 5 näher vorgestellt werden wird. Die Schülerinnen und Schüler, die im Anschluss an die Unterrichtseinheit PuMa ihr Puppenhaus den Eltern vorgestellt und die Konfigurationen erläutert haben, haben den Computer und die benutzte Software mit keinem Wort erwähnt. Der Rechner war nur Mittel zum Zweck, ihre Konfiguration geeignet darzustellen und dem System bekannt zu geben. In der Lernumgebung Kara dient der Computer hingegen nicht nur zur Darstellung der Konfiguration, sondern simuliert gleichzeitig das System mit seinen Eingaben und Ausgaben.

Reale technische Systeme zu konfigurieren lässt den Computer also sinnvoll aus dem Blickpunkt des Interesses verschwinden. Diese zweckorientierte Vorgehensweise ist Leitidee in der Technikdidaktik (vgl. dazu Hartmann, Kussmann, Scherweit 2008), der Informatik als ein Fach mit technischem Bezug entspricht.

Nach Meinung der Autorin ist für Schülerinnen und Schüler das Erzeugen *konkreter* eigener Produkte wie z. B. einer Alarmanlage im Puppenhaus (siehe Kapitel 5) oder ein Strichcodescanner mit LEGO eine wichtige Erfahrung, gerade in der Sekundarstufe I. *Denn nur durch das Sammeln vieler unterschiedlicher konkreter Erfahrungen kann sich eine Basis entwickeln, auf deren Grundlage ein Lernender später Probleme einordnen, klassifizieren und reflektieren kann.*
Für die Autorin stellen sich die Unterschiede zwischen einem Unterricht in der Sekundarstufe I und Oberstufe ganz allgemein folgendermaßen dar:



Schülerinnen und Schüler der Sekundarstufe I müssen im Informatikunterricht eigene Algorithmen entwerfen, Fehler machen und eigene Strategien entwickeln. Sie müssen selbstständig programmieren, um dann in der Oberstufe über die Eigenschaften von Algorithmen nachdenken zu können. Nur mit einem Erfahrungsschatz und der nötigen Reife zur Abstraktion und Reflexion kann der Lernende in der Oberstufe z. B. über die Komplexität oder die Korrektheit von Algorithmen reden.

Beispiel 35:
Eine in der Mittelstufe erfolgreich eingesetzte Unterrichtssequenz „Datenbanken" (Bartels 2009) sieht vor, dass die Lernenden konkrete SQL-Anfragen an eine bestehende Datenbank „Welt" stellen. Die Lernenden müssen die Berge nach ihrer Höhe ordnen, die größten Städte der Welt nach Einwohnerzahl auflisten oder die fünf längsten Flüsse anzeigen. In der Oberstufe können die Erfahrungen im Umgang mit Datenbanken und Datenbankabfragen dann in eine Modellierung einer eigenen neuen Datenbank einfließen, oder die Grundstruktur von SQL-Kommandos kann in einem Syntaxdiagramm formuliert werden.

Beispiel 36:

In der Mittelstufe erzeugen Schülerinnen und Schüler gerne eigene Geheimtexte für ihre Klassenkameraden mit Stift und Papier. Sie versuchen monoalphabetische Verschlüsselungen zu knacken, indem sie Geheimbuchstaben zählen und entsprechend den Häufigkeiten im deutschen Alphabet ersetzen. Oberstufenschüler können mit dem Erfahrungsschatz über sprachliche Redundanzen auch über polyalphabetische Verschlüsselungen nachdenken und die mathematischen Grundlagen z. B. des Friedmann-Tests durchdringen.

Beispiel 37:

Ein Mittelstufenschüler sollte *konkrete Produkte erzeugen*, z. B. eine Rechenschaltung mithilfe digitaler Elektronikbausätze zusammenstecken und sich dabei mit Kabelbrüchen und zu niedrigen Spannungen auseinandersetzen, während ein Oberstufenschüler das Abstraktionsniveau erreicht hat, z. B. programmierbare Schaltwerke zu modellieren und zu entwickeln.

Diese Trennung gilt aber nicht nur für die Sekundarstufe I und II. Auch Unterricht in der Sekundarstufe I sollte nach Meinung der Autorin entsprechend dem Schema den Schülerinnen und Schülern zunächst ohne lange Lehrervorträge und Erarbeitungsphasen konkrete Erfahrungen ermöglichen. *Dies sollte in ganz unterschiedlichen Lernumgebungen geschehen, die semantisch äquivalent sind*. Aufgrund der vielfältigen Erfahrungen sind die Schülerinnen und Schüler *anschließend* in der Lage, durch einen Vergleich der Gemeinsamkeiten der Lernumgebungen die Grundideen und Grundbausteine z. B. von Algorithmen zu identifizieren. Diese Grundlagen können dann sinnvoll systematisiert werden. Dieser Aspekt unterstützt wieder das zweckorientierte Schema. Unterrichtserfahrungen der Autorin zeigen, dass Schülerinnen und Schüler den Zweck einer Systematisierung nach einer Phase eigenen Ausprobierens als sinnvoll empfinden. Kapitel 6 beschreibt eine Unterrichtssequenz, die nach diesem Schema aufgebaut ist.

4.2.2 Fächerübergreifender Unterricht

Heymann (Heymann 1996) beleuchtet in seinem Katalog von Anforderungen an einen allgemeinbildenden Unterricht auch den Punkt „Stiftung kultureller Kohärenz“. Witten fasst den Inhalt dieses Punktes zusammen: „Neben dem Zusammenhang von Vergangenheit, Gegenwart und Zukunft sollen auch synchrone Verflechtungen innerhalb der Gesamtkultur sichtbar gemacht und Brücken zwischen einander partiell fremden Teilkulturen gebaut werden“ (Witten 2003). Es lassen sich darin zwei Teilaspekte erkennen, die für die Programmierung von Belang sind: Dem dia-chronen Aspekt, der Kontinuität zwischen Altem und Neuem, wird der Punkt gerecht, dass die

Tätigkeit der Systemkonfiguration in Kapitel 2 teilweise der fundamentalen Idee „Algorithmik“ nach Schwill (Schwill 1993) entspricht, zumindest das Horizonzal- und Vertikalkriterium erfüllt. Für die synchrone Verflechtung partiell fremder Teilkulturen gilt (bei einer fachlichen Interpretation), dass „Informatik per se fachübergreifend und fächerverbindend” ist und daher „Interdisziplinarität ein Grundsatz in der Unterrichtsgestaltung“ (Puhlmann et al. 2008) darstellt.
Auch mit der Konfiguration technischer Systeme ist fächerübergreifendes Lernen gut zu kombinieren. Am Beispiel des Blutdruckmessgerätes aus Beispiel 12, Kapitel 3 wird deutlich, dass ohne Kenntnisse aus der Biologie auch technisch gut geschulte Schülerinnen und Schüler das Gerät nicht vollständig erklären können. Wenn das Techniksystem MP3-Player konfiguriert wird, dann sind eben auch die Komprimierung einer Musikdatei und damit auch Erkenntnisse aus Musik und Biologie zu Rate zu ziehen. Viele Töne können vom Menschen überhaupt nicht unterschieden werden und können somit komprimiert gespeichert werden. Nach Meinung der Autorin soll sich fächerübergreifender Informatikunterricht dabei nicht nur auf ein Fach beziehen. Im Mittelpunkt muss das Verständnis für technische Systeme stehen, und wenn für ein besseres Verständnis die Erkenntnisse anderer Fächer benötigt werden, dann müssen die Schülerinnen und Schüler diese entweder selbst einbringen oder sich erarbeiten.
In allen Schulfächern lernen die Schülerinnen und Schüler Fakten. Jedes Schulfach bildet eine Insel des Wissens, die im Laufe der Schulzeit größer wird. Leider werden die Inseln viel zu wenig vernetzt, obwohl doch erst die Anwendung des Wissens und das Zusammenfügen der einzelnen Wissensgebiete zu etwas Neuem einen denkenden und kreativen Schüler ausmacht.

4.2.3 Projektunterricht

Wenn Fächerübergriff gewollt ist und Probleme von den Schülerinnen und Schülern gelöst werden sollen, bei denen die Programmierung als Werkzeug genutzt wird, dann eignet sich als Unterrichtsform besonders der Projektunterricht.
Die Fachliteratur kennzeichnet ein Projekt oft als größeres Vorhaben, in dem Lernende eine gemeinsam beschlossene Aufgabe praktisch, konstruktiv und eigenständig bearbeiten. Die selbstverantwortliche, lebensnahe und fächerübergreifende Struktur wird dabei besonders betont. Peterßen (Peterßen 1999) hebt die gemeinsame Lernarbeit der Schülerinnen und Schüler, das selbstständige Lernen und die Auseinandersetzung mit einem realen Problem im Gegensatz zu konstruierten Problemen hervor, wie sie sonst in der Schule üblich sind.
Projektunterricht ist immer zweckorientiert, da am Ende ein konkretes Produkt steht.
Projekte lassen außerdem viel Raum für einen differenzierten Unterricht (vgl. dazu Bönsch 2004), der jedem Lernenden unabhängig vom Leistungsstand einen persönlichen Erfolg vermitteln kann

und jeden Lernenden auf seine Weise zum Ergebnis beitragen lässt. Gerade im Informatikunterricht haben die Schülerinnen und Schüler meist unterschiedliche Vorkenntnisse, was zu sehr heterogenen Gruppen führt. Durch die Wahl eigener Projekte kann das Anforderungsniveau von den Lernenden selbst ihrem Lerntempo angepasst werden.

Projekte werden dabei von den Ideen und Visionen der Schülerinnen und Schüler gelenkt. Als Endprodukt soll nicht etwas herauskommen, was der Lehrende schon vorformuliert in seiner Schublade hat, sondern neuartige Ideen und Richtungen, die der Kreativität der Schülerinnen und Schüler entsprungen sind, sollen das Ergebnis beeinflussen. So kann ein und dasselbe Projekt in verschiedenen Lerngruppen zu unterschiedlichen Resultaten führen. Nach Ansicht der Autorin ist besonders hervorzuheben, dass eine *eigene* Problemstellung gewählt wird. Die Unterrichtserfahrung zeigt, dass Schülerinnen und Schüler, die zu einem Projekt „gezwungen“ werden, oft viel mehr Energie aufbringen, sich dagegen zu wehren, als produktiv an der Aufgabe zu arbeiten.

In Kapitel 6 werden einige Unterrichtssequenzen beschrieben. Vorweg sei gesagt, dass dort die Einführung einer neuen Lernumgebung immer einen Block beinhaltete, in dem die Schülerinnen und Schüler selbst Aufgaben (mit Lösungen) entwickelt haben. Eigenständige Konfigurationen selbst ausgedachter Aufgabenstellungen wurden den Mitschülern vorgeführt oder in Form von Aufgaben zur Verfügung gestellt. In der Unterrichtsreihe mit PuMa gab es von Anfang an nur eigene Projekte, die realisiert wurden.

4.3 Zusammenfassung

In diesem Kapitel wurde zunächst herausgearbeitet, wie wichtig die *eigenständige* Algorithmensuche bei der Konfiguration technischer Systeme ist. Legitimiert wurde die Meinung der Autorin durch Erkenntnisse aus der Mathematikdidaktik. Weiterhin wurde verdeutlicht, dass ein Unterricht im didaktischen Rahmen der Systemkonfiguration zweckorientiert und fächerübergreifend sein kann. Nach Meinung der Autorin sollten in einem Unterricht „für Alle“ in der Sekundarstufe I konkrete Produkte erzeugt werden und vor einer Systematisierung vielfältige Erfahrungen gesammelt werden.

In Kapitel 2 wurde die objektorientierte Modellierung als Konzept in der Sekundarstufe I vorgestellt. Ein weiteres Konzept (allerdings nicht explizit für die Mittelstufe) ist die „Dekonstruktion von Systemen“. In dieser Arbeit wird ein dritter Ansatz, die eigenständige Systemkonfiguration technischer Systeme, entwickelt. Folgende Tabelle stellt diese drei Ansätze nebeneinander:

Konzept	**Vorrangiges Ziel**	**Unterschied zu den anderen Konzepten**
Modellierung	Förderung der Modellierungs-kompetenzen der Lernenden	Die Ergebnisse bleiben abstrakt
Dekonstruktion von Systemen	Verstehen vorhandener Systeme und didaktischer Software aus der Lebenswelt der Schüler-innen und Schüler	Keine eigenständige Entwick-lung einer Lösung.
Systemkonfiguration	Schülerinnen und Schüler re-konstruieren und modulieren technische Systeme ihrer Lebenswelt	Schülerinnen und Schüler ent-wickeln eigene konkrete Pro-dukte.

Tabelle 6: Gegenüberstellung von Unterrichtskonzepten

5 Die PuMa-Lernumgebung – Hausautomation im Puppenhaus

In den vorhergehenden Kapiteln wurde als Konzept eines Anfangsunterrichts „Programmieren“ in der Sekundarstufe I das in dieser Arbeit entwickelte Konzept der Systemkonfiguration technischer Systeme eingeführt. In diesem Kapitel wird eine von der Autorin entwickelte Lernumgebung vorgestellt, die auf diesen Anforderungen basiert. Die Einsatzmöglichkeiten, Vorgehensweisen und Ergebnisse des Unterrichts werden dargestellt.

5.1 Haussteuerungssysteme

Zur Automatisierung im Haushalt bieten verschiedene Firmen sogenannte Haussteuerungssysteme an. Eine zentrale Einheit kann dabei vom Bediener „programmiert“ werden. Im Internet (contronics 2009) findet man: „Die HomeMatic-Zentrale wird [...] über den PC programmiert [...]“. Sensoren nehmen bestimmte Werte auf, und der Bediener steuert damit z. B. die Verriegelung von Fenstern und Türen, das automatische Öffnen von Garagen- oder Zufahrtstoren, das Herunterfahren von Jalousien u. Ä.

In einer Werbebroschüre für solch ein Haussteuerungssystem (homematic 2009) kann man lesen:

HomeMatic CCU Protokoll:

19:50 Uhr:	Signal Fernbedienung: Szene „TV-Abend“ aktivieren - Halogenlampen Decke: 50%, Stehlampen 30% dimmen - Fensterelement TV: Kippstellung 10% - Haustür verriegeln - Garagentor schließen	19:55 Uhr:	Öffnen Terrassentür
19:55 Uhr:	Betätigung Wandtaster	20:00 Uhr:	Signal Fernbedienung: Szene „Leaving-Home“ aktivieren - TV- und Audiogeräte aus - Alle Fenster schließen - Haustür entriegeln - Garagentor öffnen - Start Anwesenheitssimulation - Alarmfunktion aktivieren

Abbildung 35: Beispiel einer Homematicsteuerung, aus http://www.homematic.com/, Zugriff: 15.2.2009

Natürlich gibt es auch Sensoren, beispielsweise Feuchtesensoren, die das Schließen von Fenstern bei Regen auslösen, oder Temperatursensoren, die die Heizungsanlage steuern. Das System besteht demnach aus Sensoren (Feuchtesensoren, Temperatursensoren, Lichtsensoren, ...) und Aktoren (Motoren zum Öffnen von Toren, Lampen, ...) und einer Möglichkeit, dieses System entsprechend den eigenen Wünschen zu konfigurieren. Damit erfüllt eine Hausautomatisierung folgende Anforderungen an eine Lernumgebung im Sinne dieser Arbeit:

- Die Lernumgebung sollte sich am didaktisch-methodischen Handlungsschema Systemkonfi-

guration orientieren. Ein System muss vorhanden und intuitiv zu erfassen sein. Es sollte lediglich darum gehen, dieses System für die gewünschten Ansprüche zu spezialisieren. Dabei muss ein konkretes Produkt entstehen.

- Das System muss wie Alltagsgeräte ein technisches System darstellen, das einen Bezug zur Lebenswelt der Schülerinnen und Schüler besitzt und dem Schema: Sensor → Verarbeitung → Aktor folgt.
- Die Lernumgebung sollte eine eigenständige Suche der Konfigurationen ermöglichen.
- Bei der zu gestaltenden Lösung sollte sich etwas bewegen, etwas passieren.
- Die Aufgaben sollten selbstständig im Rahmen eines Projektunterrichts gefunden und deren Lösung erarbeitet werden.
- Die Aufgaben sollten vielfältig und in der Komplexität angemessen und unterschiedlich sein, damit differenzierter Unterricht möglich ist.

In der Programmierung von Haussteuerungssystemen reichen die in Kapitel 3 diskutierten Algorithmenbausteine Sequenz, Alternative, Schleife, disjunkte Parallelität, boolesche Variable sowie Operationen zum Lesen von Sensorwerten und Operationen zur Aktivierung/Deaktivierung von Aktoren aus.
Man könnte dieses Hausautomatisierungssystem einfach in den Klassenraum integrieren. Die Schülerinnen und Schüler würden z. B. eine Pflanzenbewässerung für die Sommerferien programmieren, die mittels Feuchte- und Temperatursensor die Bewässerung, Lüftung und den Lichteinfall für die Pflanzen optimiert. Allerdings ist diese Ausstattung zum einen sehr teuer und für Schulen kaum zu bezahlen, zum anderen kann nur jeweils eine kleine Schülergruppe diese Programme entwerfen, und das auch nur in einer Programmierumgebung, die nur wenige Eingaben und Verknüpfungen zulässt. Die Sensoren und Aktoren lassen sich ohne dieses Tool kaum ansprechen.

5.2 Das PuMa-System

Aus der Idee der Haussteuerung für möglichst viele Schülerinnen und Schüler und mit der Möglichkeit, eigene Programmieroberflächen zu kreieren, ist das PuMa-System (**Pu**ppenhaus-Hausauto**ma**tion) entstanden. Eine Schülergruppe von maximal *vier* Schülerinnen und Schülern hat jeweils ein „Puppenhaus“ bekommen. (In dieser Unterrichtsreihe waren insgesamt *acht* Häuser der Firma playmobil® im Einsatz).[16]

[16] Möglich gemacht hat diese Anschaffung die Stiftung Niedersachsen Metall, die sich für Informatik als technisches

Abbildung 36: Playmobil-Puppenhaus (Vorderansicht), Bild aus: http://www.playmobil.de, Zugriff 10.07.2009

Abbildung 37: Rückseite des Playmobil-Hauses mit einigen Verkabelungen

Als Sensoren und Aktoren wurden technische Standardbauteile der Firma Conrad (conrad 2009) verwendet. Jede Schülergruppe bekam folgende Sensoren: zwei Schalter, einen Berührsensor/ Drucksensor, zwei Lichtsensoren und einen Temperatursensor. Als Aktoren konnte jede Gruppe über einen Summer, kleine Glühbirnen, LEDs und Solarmotoren[17] verfügen.

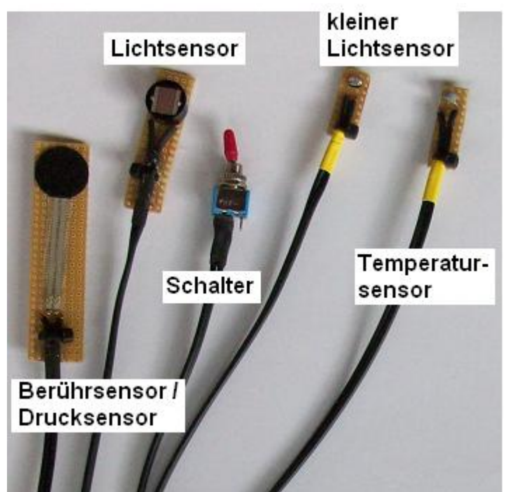


Abbildung 38: Sensoren

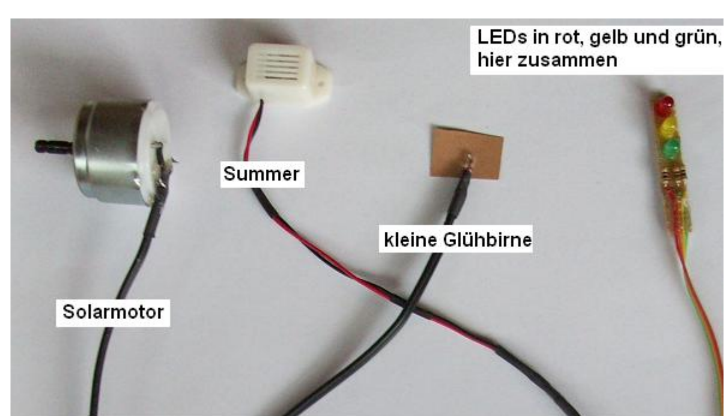


Abbildung 39: Aktoren

Das „velleman-K8055Board“ (velleman 2009) kann per USB-Schnittstelle an den Computer angeschlossen werden. Es besitzt zwei analoge und fünf digitale Eingänge sowie acht digitale und

Schulfach in der Sekundarstufe I vielfältig einsetzt.

[17] Techniker des Max-Planck-Institutes für Sonnensystemforschung in Katlenburg-Lindau haben diese Motoren freundlicherweise mit einem Legobauteil versehen, so dass die Schüler mechanische Probleme bei der Steuerung von Haustür oder Markise mit Hilfe von Legobauteilen lösen konnten.

zwei analoge Ausgänge. Eine dynamische Bibliothek (DLL-Datei) kann unter Delphi und anderen Sprachen eingebunden werden und liefert Operationen zum Setzen und Löschen der Ausgänge sowie zur Abfrage der Werte an den Eingängen.

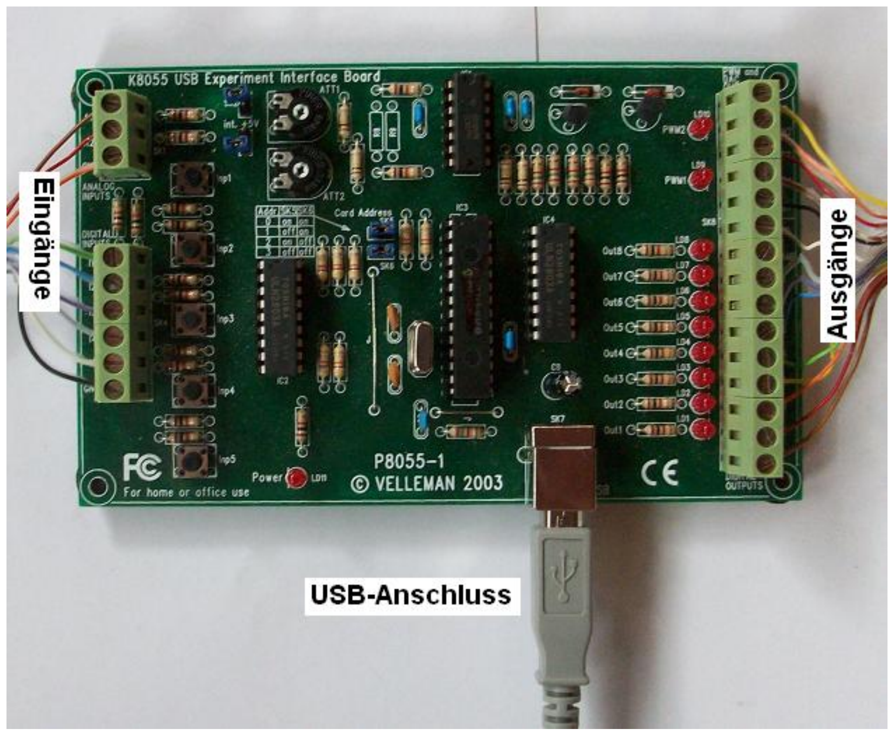


Abbildung 40: velleman-Board

Um den Fokus auf die Programmierung und nicht auf technische Schaltungen zu legen, wurden weitere Eingangs- und Ausgangsplatinen[18] gefertigt, die mit dem velleman-Board über ein Breitbandkabel verbunden sind. Die Eingangs- und Ausgangsplatinen ermöglichen den Schülerinnen und Schülern ein einfaches Anschließen der Sensoren an den Eingängen und der Aktoren an den Ausgängen, ohne sich um technische Details (insbesondere eine Spannungsverstärkung) zu kümmern. Ein weiterer Vorteil besteht darin, dass mit Hilfe von Potentiometern (siehe Abbildung 41) Einstellungen vorgenommen werden können, die auch bei Berühr-, Licht- und Temperatursensoren digitale Signale je nach Einstellung der Lernenden liefern. Das bewirkt, dass die Schülerinnen und Schüler keine Spannungswerte verarbeiten müssen, sondern selbst einstellen können, ab welcher Helligkeit z. B. der Lichtsensor den Wert true liefern soll.

[18] An dieser Stelle gebührt der Firma Solartec Dr. Witte aus Katlenburg-Lindau großer Dank.

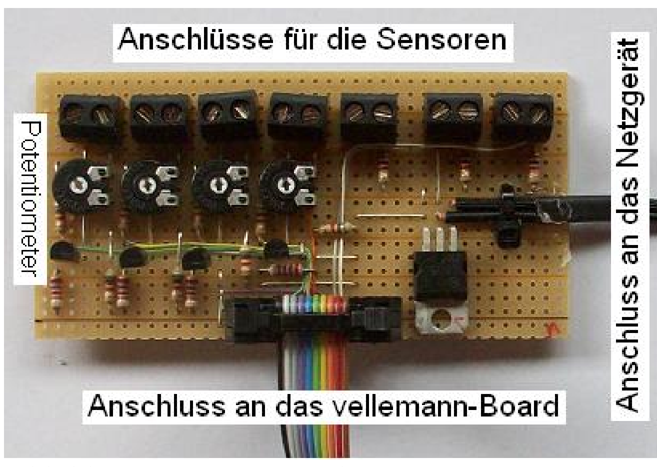


Abbildung 41: Eingangsplatine

Abbildung 42: Ausgangsplatine

Das PuMa-System besteht damit aus dem Puppenhaus mit den Sensoren und Aktoren, die die Schülerinnen und Schüler je nach Aufgabenstellung oder eigener Idee in dem Haus anbringen. Die Sensoren werden an die Eingangsplatine angeschlossen, und bei dem Wunsch nach digitalen Signalen werden die Potentiometer entsprechend eingestellt. Die Aktoren werden an die Ausgangsplatine angeschlossen. Jetzt muss dieses System konfiguriert werden. Dazu wird eine Konfiguration erstellt, die die Eingangswerte verarbeitet und Werte an die einzelnen Ausgänge gibt, die dann die Aktoren ansteuern. Diese Konfiguration wird am PC mit unterschiedlichsten Beschreibungsmitteln erstellt. Die verschiedenen Mittel zur Darstellung der Konfiguration werden am Ende dieses Kapitels näher beschrieben.

In einer durchgeführten Unterrichtseinheit waren die Schülerinnen und Schüler der Klassenstufe 8 frei in der Realisierung eigener Aufgabenstellungen. An dieser Stelle sollen exemplarisch einige Ergebnisse genannt werden.

Beispiel 38:

Schülerprodukt 1:

Ein Lichtsensor ist am Hausdach angebracht. Im Haus befinden sich Glühlampen. Wenn es dunkel wird, schaltet das Programm die Lampen an, wenn es hell ist, schaltet das Programm die Lampen aus.

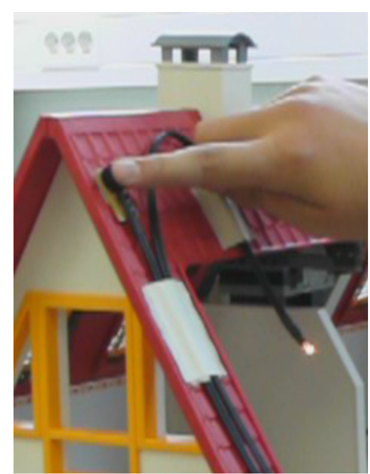


Abbildung 43: Schülerprodukt 1

Beispiel 39:

Schülerprodukt 2:

Auf der Treppe befindet sich ein Berührsensor/Drucksensor und am Fuße der Treppe ein Schalter. Mit Hilfe des Schalters kann die Alarmanlage „scharf" geschaltet werden. Geht jetzt ein Einbrecher die Treppe hoch, ertönt der Summer und eine rote LED blinkt. Ist die Alarmanlage nicht scharf geschaltet passiert nichts, wenn man auf den Berührsensor „tritt".

Beispiel 40:

Schülerprodukt 3:

Ein Schalter „aktiviert" das Aus- bzw. Einfahren der Markise. Ist er an, fährt die Markise ein oder aus, je nach Schalterstellung des zweiten Schalters.

Abbildung 44: Schülerprodukt 3 in Arbeit

Abbildung 45: Schülerprodukt 3 nach Fertigstellung

Beispiel 41:

Schülerprodukt 4:

Ein Temperatursensorwert wird vom Programm abgefragt. Wenn es zu heiß ist, wird ein Ventilator angeschaltet, der mit Hilfe eines Solarmotors realisiert wurde.

Beispiel 42:

Schülerprodukt 5:

Eine Haustür ist mit zwei Motoren und einem Taster ausgestattet. Betätigt man den Taster, dann ertönt kurze Zeit ein Signal (Summer) als Klingelzeichen. Nach einiger Zeit öffnet sich mit Hilfe eines Motors die Tür. Nach acht Sekunden (Zeit für die Familie mit Hund das Haus zu betreten), schließt sich die Tür mit Hilfe eines zweiten Motors wieder.

Abbildung 46: Haustür von außen

Abbildung 47: Haustür von innen

Die verschiedenen Konfigurationen der oben genannten Schülerprodukte wurden teilweise im selben System realisiert. Es handelt sich also um verschiedene disjunkt parallel implementierte Konfigurationen. Auf die Darstellungsform wird später eingegangen werden.

5.3 Die Evaluation

Die 20 Schülerinnen und Schüler der 8. Klasse, die an dieser Unterrichtseinheit teilgenommen haben, bewerteten die Unterrichtseinheit anschließend anonym in einem Fragebogen.

Fragebogen 1. Geschlecht: □ Junge □ Mädchen 2. Folgende Fächer interessieren mich in der Schule am meisten: ____________________	8. „Ich glaube, dass mir die Unterrichtseinheit PuMa geholfen hat, auch andere Technische Systeme im Alltag (z. B. Wäschetrockner) mit Sensoren und Aktoren besser zu verstehen“ trifft voll zu □ □ □ □ □ □ trifft überhaupt nicht zu
3. „Mir hat die Unterrichtseinheit PuMa („Hausautomation im Puppenhaus“) Spaß gemacht“ trifft voll zu □ □ □ □ □ □ trifft überhaupt nicht zu	9. „Ich glaube, dass mir die Unterrichtseinheit PuMa geholfen hat, weniger Berührungsängste mit Technischen Systemen zu haben“ trifft voll zu □ □ □ □ □ □ trifft überhaupt nicht zu
4. Am Interessantesten fand ich in der Unterrichtseinheit PuMa: ____________________	10. „Ich fand es gut, dass wir auch mit dem **scratch-Board** und den Sensoren gearbeitet haben“ trifft voll zu □ □ □ □ □ □ trifft überhaupt nicht zu
5. „Ich habe in der Unterrichtseinheit PuMa viel über Sensoren und Aktoren gelernt“ trifft voll zu □ □ □ □ □ □ trifft überhaupt nicht zu	11. „Ich würde auch anderen Lehrern und Lerngruppen zu einer Unterrichtseinheit PuMa raten“ trifft voll zu □ □ □ □ □ □ trifft überhaupt nicht zu
6. „Ich habe das Gefühl, dass ich im Bereich Technik viel dazu gelernt habe“ trifft voll zu □ □ □ □ □ □ trifft überhaupt nicht zu	12. Versuche das PuMa-System und die LEGO-Roboter gegenüber zu stellen. Wo gibt es Gemeinsamkeiten, Unterschiede? Was hat mehr Spaß gemacht? Wo sind die Vor- und die Nachteile?
7. „Ich habe das Gefühl, dass ich im Bereich Programmierung viel dazu gelernt habe“ trifft voll zu □ □ □ □ □ □ trifft überhaupt nicht zu	____________________ ____________________ ____________________ ____________________

Tabelle 7: Fragebogen zur Evaluation der Unterrichtseinheit PuMa

Einige Ergebnisse werden im Folgenden vorgestellt:

In Frage drei nach dem „Spaß“ der Unterrichtseinheit könnte man den sechs Ankreuzmöglichkeiten Schulnoten zuordnen. Dann bewerten Jungen und Mädchen die Einheit *gleichermaßen* mit der Note *gut*. Ganz offensichtlich ist eine Automatisierung von Puppenhäusern also keine Lernumgebung, bei der sich die Mädchen eher angesprochen fühlen oder die Jungen aufgrund der Puppenhäuser zurückschrecken.

Interessant ist das Ergebnis auf die Frage, was denn am meisten Gefallen gefunden hat. Die freie

Antwort wurde im Nachhinein kategorisiert in:

- Basteln (Verlegen der Kabel, Anbringen von Sensoren/Aktoren, Lösen mechanischer Probleme, ...)
- Programmierung (Angabe der Systemkonfiguration mit Hilfe des PCs)
- Erfolgserlebnis (Produktstolz, Freude, wenn etwas gelungen ist, ...)
- Alles (keine genauere Auskunft der Schülerinnen und Schüler)
- Hardware (Umgang mit den Platinen)

Es ergibt sich folgendes Bild auf die Frage: „*Was hat am meisten Spaß gemacht?*“

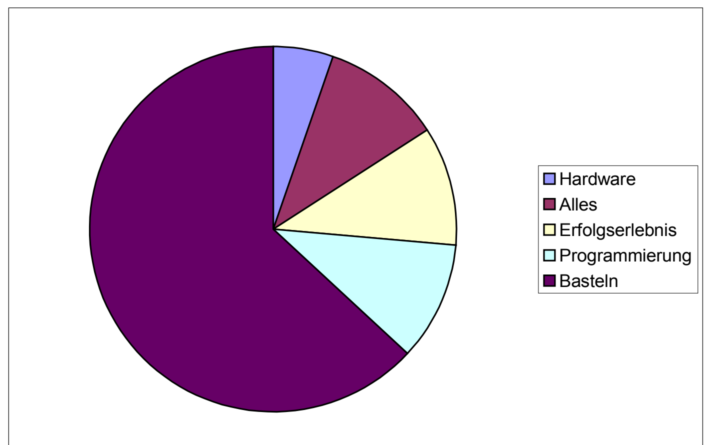


Abbildung 48: graphische Darstellung der Antworten auf die Frage „Was hat am meisten Spaß gemacht?“

Grundlage waren 20 Antworten.

In Kapitel 4.2.1 wurde der Aspekt des zweck- und produktorientierten Unterrichts hervorgehoben. Es wurde gezeigt, dass gerade Schülerinnen und Schüler der Mittelstufe konkrete eigene Produkte erstellen sollten, die sie den Mitschülern vorführen können. Der Produktstolz auf ein Ergebnis, bei dem sich etwas tut, etwas bewegt, ist in der Sekundarstufe I nach Ansicht der Autorin kaum zu überschätzen. Nach den Äußerungen der Lernenden in diesem Fragebogen muss diese Aussage noch weiter verschärft werden. An dieser Befragung sieht man, dass Schülerinnen und Schüler die Erfahrung des Arbeitens und Bastelns mit der Hand sehr schätzen. Ein weiterer Punkt, der dafür spricht, eine *reale* produktorientierte Lernumgebung wie PuMa oder das LEGO-System einer virtuellen Lernwelt wie Kara oder Karol vorzuziehen. Offensichtlich lohnt es sich, das Augenmerk auf haptische Wahrnehmungen zu richten. Das Bauen und Werken, wie es vom VDI (Verein Deutscher Ingenieure) in seinem Slogan „Sachen machen“ unterstützt wird, kommt im Gymnasium kaum vor. Ein Fach „Werken“ oder „Technik“, in dem manuell gefeilt, gesägt oder gebastelt wird, ist lediglich in den Kunstunterricht integriert.

Mit der Beschränkung ab Kapitel 3 auf die Konfiguration technischer Systeme der Lebenswelt wurde die Komplexität von Konfigurationen sinnvoll begrenzt. Unter dem Anspruch der Allgemeinbildung sollten Schülerinnen und Schüler im Unterricht technische Systeme konfigurieren, um befähigt zu werden, die erworbenen Kompetenzen so zu transferieren, so dass sie die Funktionsweise technischer Systeme ihrer Lebensumwelt algorithmisch beschreiben können.[19] In Frage 5 bis 9 dieses Fragebogens sollten die Schülerinnen und Schüler nun ihren Lernzuwachs im Bereich „Technik" dokumentieren. Dabei konnte zwischen sechs Ankreuzmöglichkeiten gewählt werden, von 5 Punkten (trifft voll zu) bis zu 0 Punkten (trifft überhaupt nicht zu). Grundlage der Auswertung sind 20 Antworten. Die Antworten wurden aufgeteilt in eine Gruppe von 13 Schülerinnen und Schüler, deren Interessen in der Schule in *informatikfremden* Fächern liegen, wie Sport, Kunst, Geschichte, Deutsch usw., und 7 Schülerinnen und Schüler, deren Interessen in *informatiknahen* Fächern wie Mathematik und Physik liegen. Eine weitere Aufspaltung nach Jungen und Mädchen gab keine Aufschlüsse, auch weil nur ein Mädchen informatiknahe Fächer als Interessenschwerpunkt angegeben hat. Es ergibt sich folgendes Bild:

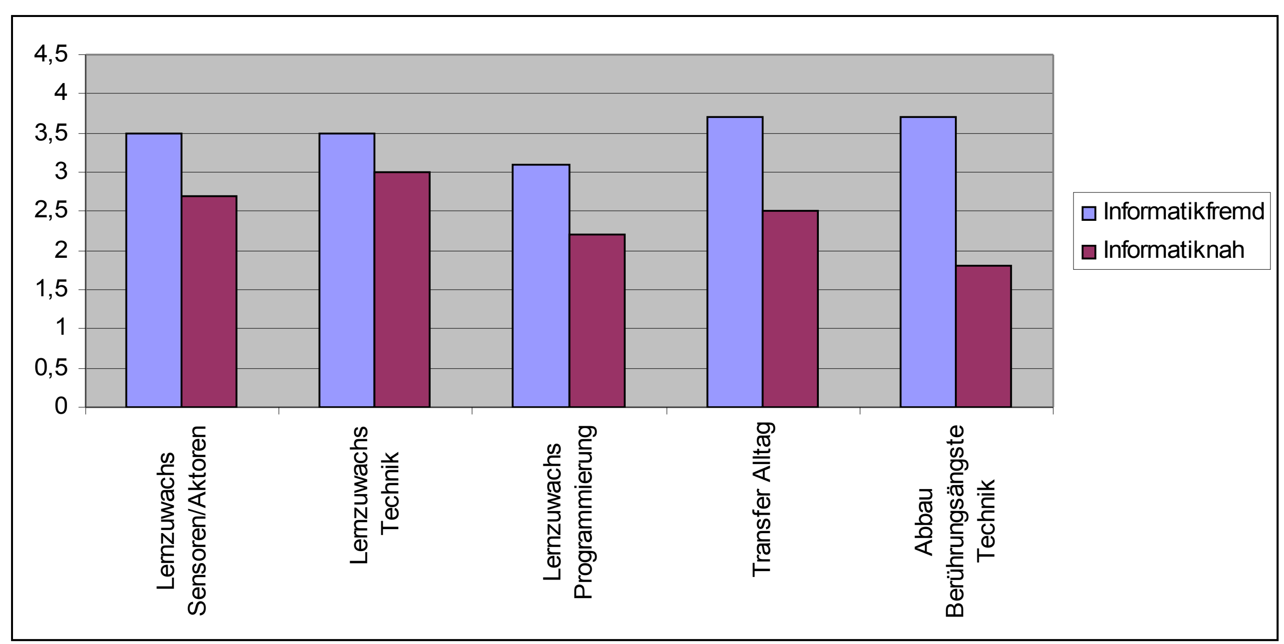


Abbildung 49: Beurteilung ihres Lernzuwachses von Schülerinnen und Schülern mit Interessenschwerpunkt in informatiknahen und informatikfremden Fächern

[19] In Kapitel 6 wird eine weitere Evaluation in einer anderen Lerngruppe vorgestellt, die dieses Ziel genauer untersucht.

Auch wenn aus statistischen Gesichtspunkten bei 20 Schülerantworten keine belastbare Aussage abgeleitet werden kann, soll doch eine Deutung versucht werden.

- Den Schülerinnen und Schülern mit Interessenschwerpunkt in informatiknahen Fächern hatten schon immer Interesse an eher technischen Dingen und sie hatten noch nie Berührungsängste mit technischen Systemen. Sie beurteilen ihren Lernzuwachs in der Unterrichtseinheit PuMa nicht so hoch wie die zweite Gruppe, weil sie auch schon vorher eine Menge Wissen in den entsprechenden Gebieten hatten. Einige dieser Schüler basteln auch in ihrer Freizeit an technischen Systemen und nehmen an Wettbewerben wie z. B. dem „Göttinger Solarcup" teil.
- Schülerinnen und Schüler, deren Interessen als informatikfremd eingestuft wurden, bewerten ihren Lernzuwachs im Bereich Technik und Programmierung deutlich höher. Sie können die Lernergebnisse auch in ihren Alltag transferieren, und ganz offensichtlich hat die Einheit PuMa zu einem Abbau von Berührungsängsten mit technischen Systemen beigetragen.

Diese Arbeit hat das Ziel, einen Ansatz im Bereich der Programmierung zu konzipieren, der *alle* Schülerinnen und Schüler erreicht, also auch gerade diejenigen, die Informatik nicht als Schwerpunkt in der Schule sehen oder besondere technische Begabungen aufweisen. Die Angaben der Schülerinnen und Schüler zeigen, dass der Ansatz dieser Arbeit mit der Lernumgebung PuMa in dieser Lerngruppe erfolgreich war. Die Ergebnisse sind so deutlich, dass es sich zu lohnen scheint, sie auf breiter Basis zu untersuchen.

5.4 PuMa und das LEGO-System

In Kapitel 3.5.2.2 wurde das LEGO-System vorgestellt. Beide Systeme eignen sich für den in dieser Arbeit entwickelten Ansatz der eigenständigen Suche einer Systemkonfiguration technischer Systeme mit dem Unterrichtsziel, Lernende zu befähigen, die Funktionsweise technischer Systeme der Lebenswelt algorithmisch beschreiben zu können. Im Einzelnen:

- Mit dem LEGO-System und PuMa können reale, konkrete Produkte erzeugt werden.
- Durch die Wahl der Beschreibungsmittel ist eine eigenständige Algorithmensuche möglich, es entstehen immer lauffähige Produkte.
- Das LEGO-System und PuMa sind technische Systeme, deren Eingaben durch Sensorwerte festgelegt sind und deren Ausgaben durch Aktoren angezeigt werden.
- Das LEGO-System und PuMa sind Systeme, die nicht modelliert werden müssen, weil sie intuitiv erfassbar sind. Sensoren und Aktoren können unmittelbar identifiziert werden.
- Es können Konfigurationen unterschiedlicher Komplexität realisiert werden.

- Die Systeme sind auf technische Systeme der Lebenswelt übertragbar. Teilweise können technische Systeme der Lebenswelt sogar rekonstruiert werden (siehe Strichcodescanner, Beispiel 14, oder Alarmanlage, Beispiel 39).

Trotzdem gibt es Unterschiede, die durch Beispiele illustriert werden sollen. Typische Aufgaben einer Unterrichtseinheit LEGO ähneln oft folgender Form:

Beispiel 43:

Der Roboter soll entlang einer schwarzen Linie fahren. Die beiden Lichtsensoren sind nun nebeneinander so angebracht, dass sie beide auf der Linie Platz haben. Der Roboter fährt geradeaus. Registriert einer der beiden Lichtsensoren weiß, dann wird der gegenüberliegende Motor für kurze Zeit abgeschaltet und der Roboter dreht entsprechend ein Stück zur Seite der schwarzen Linie.
Durch diese Aktion verändert sich der Sensorwert. Der Sensor, der eben noch weiß identifiziert hat, erkennt jetzt schwarz, und der Roboter fährt geradeaus weiter.

Die ausgelöste Aktion ist in vielen Fällen eine Bewegung, die wiederum eine Änderung der Sensorwerte bewirkt. Damit wird das LEGO-System in der Schule oft als *Regelsystem,* wie in Kapitel 3.4 beschrieben, verwendet.
Bei den Puppenhäusern (Beispiele 38 bis 42) gibt es vermehrt Aufgaben, die aufgrund aktueller oder früherer Sensorwerte Aktionen auslösen, womit meist der gewünschte (End-)Zustand erreicht ist. Diese Aktionen verändern die Sensorwerte nicht wieder. Es sind keine reaktiven Systeme. PuMa-Systeme sind nach Kapitel 3.4 *Steuerungssysteme*.
Damit ist das PuMa-System nach Meinung der Autorin etwas einfacher und sollte im Unterricht vor den Robotern Verwendung finden. Für die Schülerinnen und Schüler sind Aufgaben im PuMa-System zunächst einfacher in der Realisierung, weil man keine Wiederholungen und damit auch keine Terminierungsbedingungen braucht. Die Arbeit mit dem LEGO-Robotersystem sollte sich nach Ansicht der Autorin anschließen.

Ein weiterer Unterschied liegt in der Vielfältigkeit der Programmierumgebung. Im PuMa-System gibt es unterschiedliche Beschreibungsmittel zur Konfiguration, worauf Kapitel 5.5 näher eingeht.
Außerdem ist bei PuMa der Fluss des Informationsdurchlaufs offensichtlicher. Sensoren sind über Eingabe-Platinen an das velleman-Board angeschlossen und per USB-Anschluss mit dem Computer verbunden. Mit diesem wird die Konfiguration darstellt. Velleman-Ausgaben werden an die Ausgangsplatine und dort an die Aktoren weitergegeben. Im LEGO-System wird das Programm vom Rechner auf das System übertragen, Sensoren und Aktoren werden angeschlossen, aber durch

die kompakte Bauweise der meisten LEGO-Roboter ist der Informationsfluss durch das System nicht so offensichtlich.

Die befragten Schülerinnen und Schüler sehen die beiden Systeme weitgehend als gleichwertig an.

Abbildung 50: Schülermeinung zum Unterschied LEGO und PuMa

„Ich finde, dass es gar keine großen Unterschiede zwischen den beiden Sachen gibt. Es macht jedoch beides Spaß. Allerdings ist es schöner ein Puppenhaus zu bauen, da man dort mehr eigene Ideen einbauen kann, was man nicht so gut bei den Lego-Robotern kann."

Abbildung 51: Schülermeinung zum Unterschied LEGO und PuMa

„Bei den Lego-Robotern ging es schneller, weil die Teile bereits vorgefertigt waren. Dafür hat man bei PuMa mehr über die elektrische Schaltung gelernt, und wie die Sensoren ausgelesen werden."

5.5 Die Beschreibung der Konfigurationen in PuMa

Die Schnittstelle des PuMa-Systems, über die eine Konfiguration erfolgt, ist das velleman-Board. Hier gelangen die Eingabewerte hinein und werden zum Rechner übertragen. Ausgabewerte werden vom Rechner an die Ausgänge gelegt. Eine mitgelieferte DLL stellt die notwendigen Befehle zum Lesen der Eingabewerte und Schreiben der Ausgabewerte bereit. Es können also beliebige Oberflächen programmiert werden, die den Schülerinnen und Schülern Beschreibungsmittel zur Verfügung stellen, ihre Konfigurationen anzugeben. Das PuMa-System sollte konfigurierbar sein:

- durch eine Angabe von Schaltnetzen bzw. Schaltwerken,

- durch die Angabe einer Mealy-Maschine,
- durch eine graphische imperative Programmiersprache mit den Basisbefehlen `SetzeAusgang(Nummer, Wert)` und `LiesEingang(Nummer)`,
- durch die Angabe eines UML-Aktivitätdiagramms,
- durch die Angabe von Delphi-Codes.

Kapitel 6 zeigt Beispiele für diese Oberflächen und die *Notwendigkeit* so vieler unterschiedlicher Programmierumgebungen.

Zum aktuellen Zeitpunkt sind zwei Oberflächen lauffähig, an den anderen wird gearbeitet.

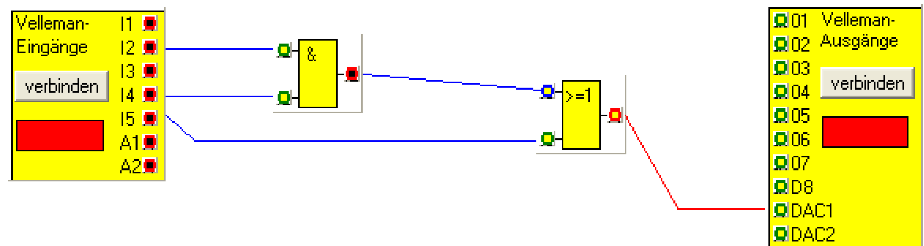


Abbildung 52: Der Hardwaresimulator HASI von Eckart Modrow (Modrow 2004) mit PuMa-Erweiterung

In den in Delphi programmierten Hardwaresimulator „HASI“ von Eckart Modrow (Modrow 2004) wurde die velleman-DLL integriert, mit deren Hilfe die Eingaben und Anzeigen der Schaltnetze und Schaltwerke auch von außen kommen können und nach außen gehen.

Die zweite lauffähige Umgebung ist Delphi selbst oder andere Programmiersprachen (C++, Java, ...). Die Schülerinnen und Schüler bekommen ein lauffähiges Delphi-Programm zur weiteren Bearbeitung, in dem die Verbindung zur Platine und die Einbindung der DLL-Datei bereits implementiert sind. Weiterhin werden mit Hilfe eines Timers alle 100 ms die Werte der Eingänge ausgelesen und in Variablen gespeichert. Dies ist für die Schülerinnen und Schüler nicht sichtbar. Sie müssen lediglich ihre eigene Konfiguration in kurze Alternativen und eventuell Variablenzuweisungen übersetzen.

Beispiel 44:

Ein einfaches Programmkonstrukt sah folgendermaßen aus:

```
if (ReadDigitalChannel(1)=true) then SetDigitalChannel(1)
 else ClearDigitalChannel(1);
```

Um die fehlenden weiteren Oberflächen in einer ersten Unterrichtseinheit auszugleichen, wurde PuMa auch mit der graphischen Programmierumgebung „scratch“ konfiguriert. Der nächste Abschnitt widmet sich dieser Konstruktion.

5.5.1 Das Puma-System und „scratch“

„Scratch“ ist eine graphische Programmierumgebung (scratch 2009). Es handelt sich um eine objektorientierte Programmiersprache mit imperativem Kern, der allein für die Konfigurationen von Bedeutung ist. „Scratch is a new programming language that makes it easy to create your own interactive stories, animations, games, music, and art – and share your creations on the web. Scratch is designed to help young people (ages 8 and up) develop 21st century learning skills. As they create and share Scratch projects, young people learn important mathematical and computational ideas, while also learning to think creatively, reason systematically, and work collaboratively“ (scratch 2009).

In Kapitel 3 wurde ein Beispiel zu „scratch“ gegeben. Sind das PuMa-Haus und die Sensoren installiert, dann kann man folgende Variante wählen: Die Sensoren werden an eine Platine, das sogenannte „Picoboard“, angeschlossen. Dieses besitzt jeweils einen integrierten Geräusch-, Licht- und Tastsensor sowie vier weitere Anschlüsse für andere Sensoren. Das Picoboard ist über einen USB-Anschluss mit dem Rechner verbunden und lässt sich automatisch in die Programmierumgebung „scratch“ integrieren. In dieser beschreiben die Schülerinnen und Schüler dann ihre Konfiguration.

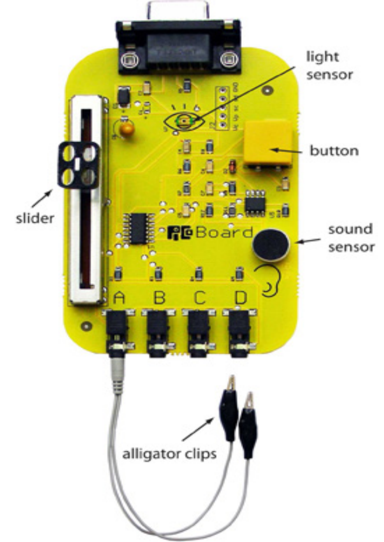


Abbildung 53: „Picoboard“

Die Verwendung von „scratch“ im PuMa-Projekt hat viele Vorteile, die den Anforderungen aus den vorhergehenden Kapiteln entsprechen. Allerdings gibt es auch Nachteile (s. u.).

Vorteile:

- „Scratch“ ist eine graphische Programmiersprache, die erlaubt, das Programm aus einzelnen Bausteinen nach dem Baukastenprinzip zusammenzufügen. Hierbei werden elementare Funktionen, z. B. die Bildschirmausgaben, optisch von Elementen zur Steuerung des Programmablaufs, z. B. Schleifen oder Alternativen getrennt .
- Syntaxfehler können nicht provoziert werden, da sich einzelne Elemente nur bei korrekter Syntax einfügen lassen.

- „Scratch“ ist eine freie Umgebung, die unkompliziert in der Installation ist und somit an allen Schulen verwendet werden kann.

Nachteile:

- Für Konfigurationsaufgaben müssen Variablen als eine Art „flag“ verwendet werden, damit bestimmte Systemzustände unterschieden werden können. „Scratch“ erlaubt (in der verwendeten Version) nur Variablen des Typs Gleitpunktzahl.
- „Scratch“ ist eigentlich viel zu mächtig für diese Art von Aufgabenstellungen. Eine abgespeckte Variante würde eine vereinfachte Bedienung ermöglichen. Allerdings kann man so in späteren Programmiereinheiten an eine bekannte Umgebung anknüpfen.
- Die Visualisierung der Alternative ist im Zusammenhang mit Schleifen unglücklich gewählt. Schließt sich an eine Programmiereinheit mit „scratch“ eine Systematisierung beispielsweise mit Struktogrammen an, dann könnten Probleme entstehen.
- Während des Programmablaufs ist in der verwendeten Version nicht ersichtlich, bei welchem Kommando sich die Ausführung gerade befindet.
- Das „Picoboard“ besitzt leider keine Ausgabemöglichkeiten. Hier liegt der Hauptnachteil in der Verwendung beim PuMa-System. Da sich etwas bewegen oder etwas passieren soll, reicht eine Bildschirmausgabe nicht aus.

Beispiel 45:

An das Picoboard ist an Eingang B der Lichtsensor angeschlossen, der einen Wert zwischen 0 und 100 liefert. Jetzt wurde von Schülern z. B. folgendes scratch-Programm geschrieben:

Abbildung 54: „scratch“-Programm mit „Picoboard“

Als Ausgabe dient die Bildschirmnachricht: „es ist dunkel“ bzw. „es ist hell“. Weiterhin bewegt sich das Objekt auf der scratch-Bühne entsprechend dem Sensorwert nach oben oder unten.

Das Fehlen einer Ansteuerungsmöglichkeit für Aktoren wird hier deutlich. Außerdem liefern Sensoren hier keine digitalen Signale. Der Vorteil ist aber, eine freie graphische Programmier-

umgebung mit der Möglichkeit des Anschlusses externer Sensoren zu haben. In Kapitel 6 wird gezeigt, dass ein Umgang mit „scratch“ im PuMa-System, wenn „scratch“ bereits bekannt ist, den Übergang zu textbasierten Sprachen erleichtern kann.

Weiterhin zeigt „scratch“, wie man sich eine graphische imperative Programmierumgebung vorzustellen hat, in der das PuMa-System konfiguriert werden kann.

5.5.2 Konfigurationsumgebungen in PuMa, die noch zu realisieren sind

Möglich und denkbar sind auch die noch nicht realisierten Programmierumgebungen, da nur Compiler geschrieben werden müssen, die die DLL-Befehle nutzen und den Schülerinnen und Schülern eine Oberfläche zur Konfiguration des PuMa-Systems zur Verfügung stellen.

Gegeben sei eine Aufgabe wie folgt:

Eine Lampe kann von einem Taster aus bedient werden. Wird der Taster betätigt, geht die Lampe an oder aus, je nachdem, ob sie vorher brannte oder nicht.

Sind Taster und Lampe im Haus installiert, sollte ein und dieselbe Konfiguration in den unterschiedlichen Beschreibungssprachen folgendermaßen aussehen:

1. Konfiguration als UML-Aktivitätsdiagramm:

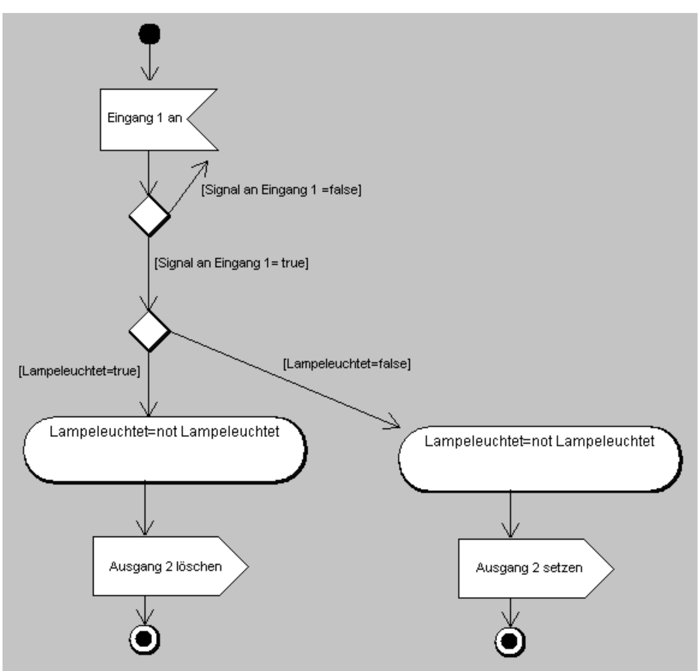


Abbildung 55: Konfiguration als UML-Aktivitätsdiagramm

2. Konfiguration als Schaltwerk:

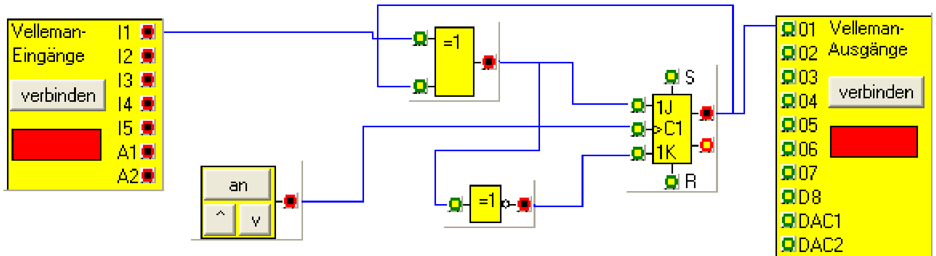


Abbildung 56: Konfiguration als Schaltwerk

3. Konfiguration als Mealy-Maschine:

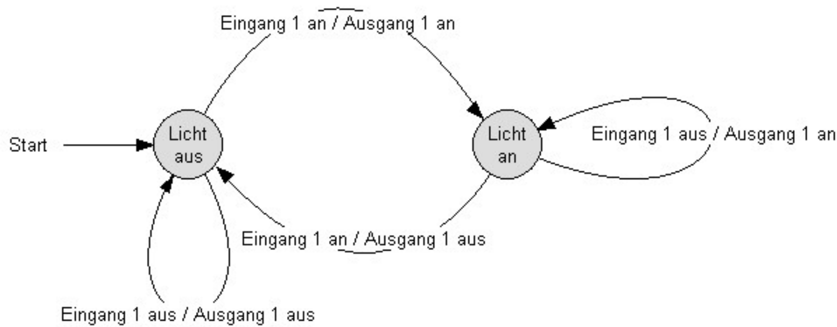


Abbildung 57: Konfiguration als Mealy-Maschine

4. Konfiguration mit einer graphischen imperativen Programmierumgebung:

Wenn Signal an Eingang 1
falls Sensor Taster gedrückt ?
falls LampeLeuchtet = true
setze Ausgang 2 auf false
sonst
setze Ausgang 2 auf true

Abbildung 58: Konfiguration als graphische imperative Anweisungsfolge (in Anlehnung an „scratch“)

5. Konfiguration als Delphi-Code (siehe Kapitel 6)

```
if (ReadDigitalChannel(1)=true) then
    begin
        if (LampeLeuchtet) then
           ClearDigitalChannel(1)
        else SetDigitalChannel(1);
        LampeLeuchtet:=not LampeLeuchtet;
    end;
```

5.6 Kategorisierung der Aufgaben im PuMa-System

In diesem Abschnitt werden die Konfigurationsaufgaben im PuMa-System klassifiziert:

Stufe 1:

Hier werden alle Aufgaben zusammengefasst, die mit Ausgaben auf bestimmte Kombinationen von Eingaben reagieren, ohne dafür zurückliegende Eingaben betrachten zu müssen oder Werte und Informationen speichern zu müssen.

Beispiel 46:

- Wenn der Temperatursensor einen bestimmten Wert liefert, wird der Ventilator angeschaltet.
- Bei einem bestimmten Druck des Berührsensors und gleichzeitiger Dunkelheit schaltet sich die Alarmanlage (Summer) an.
- Tagsüber, wenn es hell ist und die Temperatur einen bestimmten Wert überschreitet, fährt die Markise aus.

Alle diese Aufgaben können ohne Variablen gelöst werden. Sind die Eingänge und Ausgänge gesteckt, dann ergibt sich aus dem ersten Beispiel eine Konfiguration wie folgt:

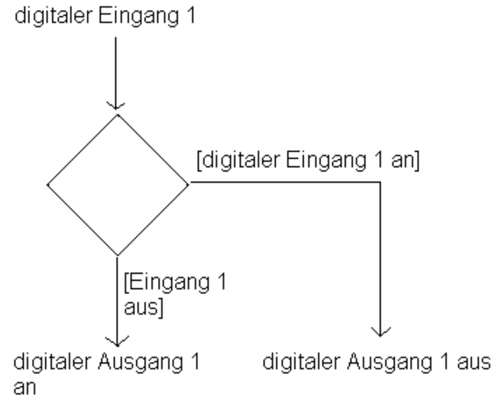


Abbildung 59: Konfiguration der 1. Stufe

In der Delphi-Programmierung findet man hier einfache if-Anweisungen:

```
if (ReadDigitalChannel(1) = false) then SetDigitalChannel(2) else ClearDigitalChannel(2);
```

Im PuMa-System kann man dank der Platinen die Sensorwerte als digitale Eingaben verwenden und die Aktoren als digitale Ausgaben. Dann könnte eine Aufgabe dieser Kategorie auch durch ein einfaches Schaltnetz mit den Grundgattern UND, ODER und NICHT programmiert werden:

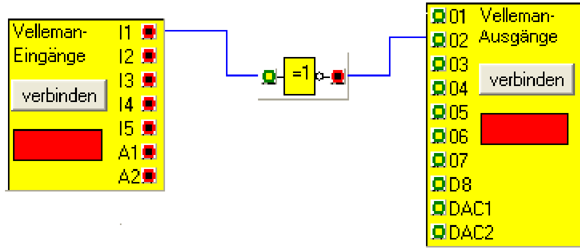


Abbildung 60: Konfiguration der 1. Stufe

Eine Mealy-Maschine mit der Möglichkeit digitaler Ausgaben und einem Alphabet, welches aus den Werten der Eingänge besteht, kann konstruiert werden und während der Laufzeit von außen Eingaben einlesen. Die Struktur der Aufgabenstellungen in dieser 1. Kategorie würde stets Mealy-Maschinen mit einem *einzigen* Zustand erfordern, da keine Informationen zu „speichern" sind. Das oben angegebene Beispiel würde sich wie folgt darstellen:

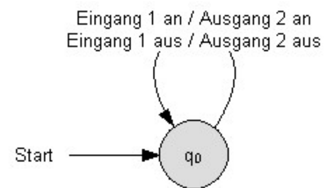


Abbildung 61:
Konfiguration der 1.Stufe

Stufe 2:

In dieser Stufe sollen alle Aufgaben gesammelt werden, die zwischen verschiedenen Systemzuständen unterscheiden müssen. Dabei werden zusätzlich zu den Operationen der ersten Stufe sogenannte „Merker" als Art „flag" benötigt, die vorherige Ereignisse „speichern" können.

Beispiel 47:

- Eine Lampe kann von einem Taster aus geschaltet werden. Wird der Taster betätigt, geht die Lampe an oder aus. Eine Betätigung des Tasters löst also eine Änderung des Zustands der Lampe aus. Es muss eine Möglichkeit geben, auf der Programmseite zu speichern, ob die Lampe brennt oder nicht.
- Die Betätigung eines Tasters aktiviert eine Alarmanlage. Wird jetzt der Berührsensor ausgelöst, ertönt der Summer. Ohne Aktivierung der Alarmanlage geht das Licht an, wenn der Berührsensor ausgelöst wird.
- Ein Taster an der Wand fährt die Markise ein oder aus, je nachdem, ob die Markise draußen oder drinnen ist.

Der Unterschied zu der 1. Stufe ist darin zu sehen, dass in der Delphi-Programmierung oder generell in einer textbasierten Sprache eine Variable eingeführt werden muss (möglichst vom Datentyp Boolean), die gewisse Informationen des Systems speichert.

Sind wieder alle Ein- und Ausgänge gesteckt, dann ergibt sich aus dem ersten Beispiel ein Algorithmus wie bereits in Kapitel 3 beschrieben:

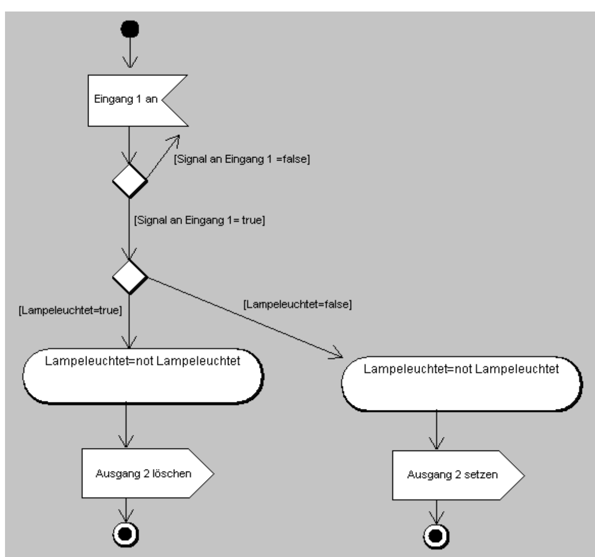


Abbildung 62: Konfiguration der 2. Stufe

Und in der Delphi-Programmierung:

```
if (ReadDigitalChannel(1)=true) then
    begin
        if (LampeLeuchtet) then
           ClearDigitalChannel(1)
        else SetDigitalChannel(1);
        LampeLeuchtet:=not LampeLeuchtet;
    end;
```

Analog zu der Programmierung mit Schaltnetzen muss in dieser Kategorie auf Schaltwerke umgestiegen werden. Dasselbe Beispiel würde folgendermaßen programmiert:

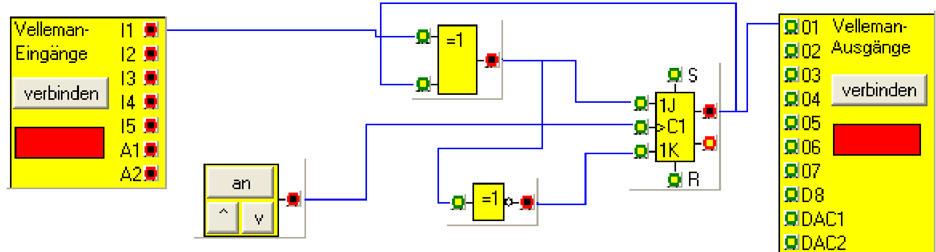


Abbildung 63: Konfiguration der 2. Stufe

Man erkennt das Flip-Flop als Speicherbaustein zur Speicherung des aktuellen Lampenzustands. Eine Mealy-Maschine hätte demzufolge mehrere Zustände, in diesem Fall zwei:

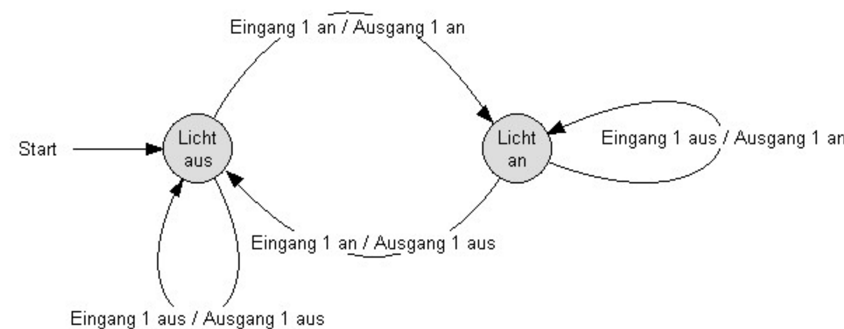


Abbildung 64: Konfiguration der 2. Stufe

3. Stufe

In die 3. Stufe der Kategorisierung der Programmieraufgaben würden all jene Probleme fallen, die eine besondere Struktur der Variablen erfordern. Wenn bestimmte Datenstrukturen wie Reihungen, Bäume oder Listen notwendig wären oder aber die Anzahl der Zustände ein überschaubares Maß überstiege, dann wären das Probleme der Kategorie 3. Offensichtlich fallen diese Art Aufgaben aber nicht in den Anforderungsbereich, der in Kapitel 3 herausgearbeitet wurde.

Kapitel 3 hat gezeigt, dass technische Systeme der Lebenswelt mit Hilfe der SPS-Programmierung gesteuert werden. Demzufolge reichen Schaltwerke oder Zustandsgraphen (Mealy-Maschinen) für die Programmierung aus. Die Klasse dieser Aufgaben stimmt genau mit der Klasse der Aufgaben der 1. und 2. Stufe nach obigem Klassifikationsschema überein. Nach Meinung der Autorin ist dies

eine angemessene Begrenzung für einen Programmierunterricht „für Alle" der Sekundarstufe I.
Die Unterteilung in die beiden ersten Stufen wird auch in anderen Unterrichtsbeispielen und Materialien so vollzogen. In Schulbüchern und auch weiterführender Literatur (Schiffmann, Schmitz 2003) zum Themengebiet „Technische Informatik" werden zunächst die Schaltnetze und dann darauf aufbauend die Schaltwerke behandelt. Materialien zur Unterrichtseinheit Automaten-Kara beginnen mit Aufgaben, die mit einem Zustand gelöst werden können. Erst später werden mehrere Zustände betrachtet.[20]

Die Nutzung unterschiedlicher Oberflächen zur Darstellung der Konfigurationen quer durch die Gebiete der Informatik hat den Vorteil, dass Schülerinnen und Schülern nicht nur verschiedene Möglichkeiten angeboten werden, die sie ihrem Lerntyp entsprechend auswählen können, sondern die Vielfältigkeit der Beschreibungsmittel hat zwei wesentliche Vorteile:

- Gebiete der Informatik können besser in Beziehung zueinander gesetzt werden. Geht man anschließend weiter auf die Einsatzgebiete und Möglichkeiten endlicher Automaten oder Schaltnetze/-werke ein, dann erreicht man eine Verknüpfung der unterschiedlichen Wissensgebiete der Informatik, die in der Sekundarstufe I bislang kaum gegeben ist. Die Schülerinnen und Schüler werden bisher in „technischer" oder „theoretischer Informatik" unterrichtet, sehen die Gebiete aber nach Ansicht der Autorin isoliert. Dokumentationen von Unterrichtseinheiten, die die Informatikgebiete in der Sekundarstufe I miteinander verbinden, gibt es nach Kenntnisstand der Autorin kaum.
- Das PuMa-System kann in verschiedenen Gebieten einen Einstieg darstellen. In der theoretischen Informatik können die Syntax von Automaten gezeigt und außerdem Grenzen der Aufgaben mit einem Zustand erarbeitet werden. In der technischen Informatik können ebenfalls Schaltnetze und Schaltwerke kreiert werden, die konkrete Produkte erzeugen und Ausgangspunkt für weitere Überlegungen sind.

5.7 Zusammenfassung

Das vorgestellte PuMa-System erfüllt die Anforderungen der Kapitel 2 bis 4. Ein reales, der Lebenswelt der Schülerinnen und Schüler nahes, technisches System mit Sensoren und Aktoren kann mit vielfältigen graphischen Beschreibungsmitteln der Systemkonfiguration quer durch alle Gebiete der Informatik konfiguriert werden. Diese Konfigurationen können aufgrund der

[20] Im Unterricht der 7. Klasse stellt der Umstieg von Aufgaben mit einem Zustand zu Aufgaben mit mehreren Zuständen nach Erfahrung der Autorin einen Bruch dar, den die Schülerinnen und Schüler oft nur mit Hilfe bewältigen.

Vielfältigkeit der Aufgaben von den Lernenden eigenständig gefunden werden und führen immer zu konkreten, lauffähigen Produkten. Mit dem PuMa-System ist die Aufgabenkomplexität, wie bei technischen Systemen generell, auf ein Maß beschränkt, das nach Meinung der Autorin für einen Unterricht „für Alle“ der Sekundarstufe I angemessen ist.

6 Unterrichtssequenzen

Im ersten Teil dieser Arbeit ging es um didaktische Reduktionen, inhaltliche Anforderungen und methodische Zielrichtungen, die zusammen eine Lernumgebung oder einen Lernraum beschrieben haben, in dem ein Programmierunterricht „für Alle“ stattfinden soll. Es ging um das, *was* im Unterricht eingesetzt werden kann, nämlich intuitiv zu erfassende Systeme mit einem Bezug zur technischen Lebensumwelt der Schülerinnen und Schüler, die eigenständig entsprechend den Aufgabenstellungen konfiguriert werden sollen. Als Möglichkeit wurde das PuMa-System vorgestellt; LEGO-Roboter, Automaten-Kara oder Karol dienen m. E. demselben Zweck. In diesem Abschnitt soll genauer darauf eingegangen werden, *wie* man mit Hilfe dieser Systeme und Lernumgebungen dem Ziel näher kommt, Schülerinnen und Schüler zu befähigen, die Funktionalität technischer Systeme algorithmisch zu beschreiben. Dazu werden zwei durchgeführte Unterrichtssequenzen angegeben und ein aus dem Unterricht der Autorin folgender möglicher Weg, die gemachten Erfahrungen so zu systematisieren, dass sie von den Lernenden verallgemeinert und auf unbekannte Systeme transferiert werden können.

Das Ziel eines Programmierunterrichts darf es nicht sein, Spezialisten bestimmter Systeme auszubilden. Ziel muss sein, Grundstrukturen zu erkennen und auf beliebige (ähnliche) Systeme zu übertragen. Ein Programmierer, der z. B. vor eine Aufgabe gestellt wird, ein Delphi-Programm zu erweitern, und sich das nicht zutraut, weil er bisher nur in C++ programmiert hat, der kann nach Meinung der Autorin nicht programmieren. Hat man die Grundstruktur eines Algorithmus oder einer imperativen Sprache verstanden, so bleibt lediglich ein Syntaxproblem zurück, das sich binnen kürzester Zeit lösen lässt. Dieses Ziel muss auch in der Schule verfolgt werden. Die Schülerinnen und Schüler dürfen nicht zu Spezialisten in Automaten-Kara ausgebildet werden, sondern sollen ihre Erfahrungen mit Automaten-Kara und anderen Lernumgebungen nutzen, um Erfahrungen so zu strukturieren, dass sie Grundprinzipien erkennen. Dazu werden zwei mögliche Wege aufgezeigt:

- das Erstellen von Konfigurationen in verschiedenen vorhandenen Lernumgebungen mit anschließender Systematisierung
- das Erstellen von Konfigurationen im PuMa-System mit unterschiedlichen Oberflächen und anschließender Systematisierung

6.1 Konfigurieren in verschiedenen Lernumgebungen mit einer anschließenden Systematisierung

In diesem Kapitel wird eine Unterrichtssequenz in Informatik einer 7. Klasse, beschrieben, die im Schuljahr 2007/2008 durchgeführt wurde. In einer kurzen Schilderung der Rahmenbedingungen wird deutlich, dass es sich um einen „Informatikunterricht für Alle" handelt, in dem der erste Kontakt mit „Programmierung" erfolgt.

6.1.1 Die Unterrichtssequenz im Schuljahr 2007/2008

Die Lerngruppe dieser Unterrichtssequenz, ein Informatikkurs des Jahrgangs 7, wurde im Wahlpflichtunterricht (WPU) nach der Niedersächsischen Stundentafel 1 in Informatik unterrichtet. Jeder WPU-Kurs setzt sich aus Schülerinnen und Schülern aller Klassen eines Jahrgangs zusammen, die wiederum auch nur in diesem Kurs in dieser Zusammensetzung unterrichtet werden. An diesem Gymnasium wählen die Lernenden im Wahlpflichtunterricht gewöhnlich eine dritte Fremdsprache. So kommt es, dass Informatik bei manchen leider auch als Ausweichfach gesehen wird, wenn das Erlernen einer dritten Fremdsprache zu schwierig erscheint. Informatik hat bei vielen Eltern sowie Schülerinnen und Schülern noch den Ruf als Fach, in dem eine Schulung an Microsoft-Office-Produkten durchgeführt wird. Dadurch war dieser Kurs mit 28 Lernenden sehr heterogen zusammengesetzt und genügt damit den Anforderungen, dass nicht nur Schülerinnen und Schüler unterrichtet werden sollen, die sich in besonderem Maße, auch über die Inhalte der Schule hinaus, für Informatik begeistern.

6.1.1.1 Übersicht des Unterrichtsverlaufs

Folgende Inhalte mit Bezug zur Programmierung wurden unterrichtet:

Inhalt	**Anzahl Stunden**
Programmierung mit Automaten-Kara	7
Programmierung: LEGO-Roboter	8
Systematisierung. Vergleich Kara und LEGO, Gemeinsamkeiten und Unterschiede, Struktogramme als Kommunikationsmittel in beiden Umgebungen	2
Evaluation	
Programmierung: LEGO-Roboter. Eigene Ideen verwirklichen, vorführen und das Programm auch als Struktogramm abgeben	5
LEGO-Projekt: Barcodescanner	2
Struktogramme für technische Systeme der Lebenswelt	4

Lernumgebung: Karol der Roboter; Gemeinsamkeiten, Unterschiede, inkl. eigene Aufgaben erfinden, Arbeit mit Struktogrammen	5

An jede Einheit schloss sich eine Zeitspanne an, in der die Schülerinnen und Schüler eigene Aufgaben für sich und ihre Mitschüler erfinden mussten. Jede Gruppe musste eine Lösung dieser Aufgabe bereits programmiert haben, dann wurden die Aufgaben der einzelnen Gruppen verteilt und von den anderen Schülerinnen und Schülern bearbeitet. Eigene Projekte, insbesondere im Zusammenhang mit LEGO oder auch die Lösung von Kara-Aufgaben, wurden stets von den Lernenden präsentiert. Die Klasse hatte sich nach kurzer Zeit an diese Arbeitsform der Erstellung eigener Produkte und deren Präsentation gewöhnt.

6.1.1.2 Die Programmierumgebungen

Automaten-Kara und das LEGO-RIS-System wurden in Kapitel 3 bereits vorgestellt. Die Lernumgebung Robot Karol ähnelt Kara sehr stark, mit dem Unterschied, dass die Lernenden die Konfigurationen textbasiert angeben müssen (siehe dazu Karol 2009). Tabelle 8 gibt nochmals eine Übersicht.

	Automaten-Kara	**LEGO-Mindstorms RIS**	**Robot Karol**
Ist es eine graphische Programmiersprache?	Ja, programmiert wird mit Hilfe endlicher Automaten.	Ja, imperative, nebenläufige Syntax.	Nein, textbasiert mit beschränktem Sprachumfang.
Basiert die Umgebung auf dem Sensor-Aktor-Prinzip?	Ja.	Ja.	Ja. Es gibt Sensoren, z. B. Wand vorne, und Aktionen, z. B. Schritt nach vorne.
Einfache intuitive Syntax?	Ja. Keine Syntaxfehler möglich.	Ja. Keine Syntaxfehler möglich.	Ja. Syntaxfehler sind möglich.
Orientierung am Handlungsschema Systemkonfiguration mit einem zugrunde liegenden System, welches intuitiv zu erfassen ist?	Ja.	Ja.	Ja. Karol lebt in einer Mini-Welt, die durch Wände beschränkt ist. Er kann Ziegel hinlegen, vorwärts gehen, die Ziegel aufnehmen, Marken setzten usw. und durch Sensoren seine Umwelt erkennen.

Ist das zugrunde liegende System auf Alltagsgeräte übertragbar? Bezug zu technischen Systemen der Lebenswelt der Schüler?	Nein, nicht unmittelbar.	Ja, ohne Einschränkungen, siehe Strichcodescanner.	Nein, nicht unmittelbar.
Imperative sequentielle Programmiersprache, die nach dem Baukastenprinzip zusammengesetzt werden kann?	Ja.	Ja.	Ja. Zunächst ist dieses Baukastenprinzip nicht so deutlich wahrzunehmen. Durch die Anzeige des zugehörigen Struktogramms aber schon.
Kann der Programmablauf verfolgt und so eine Fehlersuche vereinfacht werden?	Ja, sehr gut. Der gerade ausgeführte Befehl wird farbig unterlegt.	Nein.	Nein.

Tabelle 8: Übersicht Automaten-Kara, LEGO und Karol

6.1.1.3 Programmiereinheiten: Durchführung und Ergebnisse

1. Kara

Vorgehensweise:

Automaten-Kara war die erste Lernumgebung, mit der die Schülerinnen und Schüler Kontakt hatten. In einer etwa zehnminütigen Demonstration wurde der Umgang mit Kara erläutert und eine Aufgabe gemeinsam bearbeitet. Danach haben die Schülerinnen und Schüler selbstständig mehrere Aufgaben erfolgreich gelöst, wobei jede Gruppe in individuellem Tempo vorgegangen ist. Am Ende stand die Aufforderung an die Schülerinnen und Schüler, selbst Aufgaben für die Mitschüler zu kreieren und dann die Aufgaben der Mitschüler zu lösen.

Reflexionen:

Schwierigkeiten gab es, als die Aufgaben einen höheren Komplexitätsgrad erreichten. Die ersten Aufgaben waren von folgender Form:

Beispiel 48:

Gegeben ist folgende Welt:

Abbildung 65: Anfangsszenario

Kara soll bis zum Baum laufen, alle Kleeblätter auf dem Weg aufnehmen und vor dem Baum halten. Die Programmierung sieht wie folgt aus:

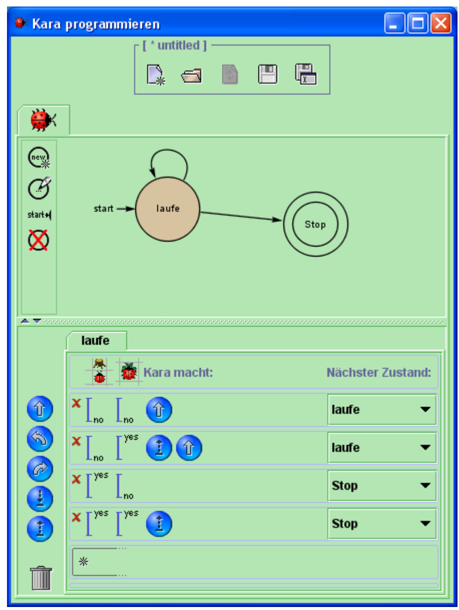


Abbildung 66: Kara-Programm der Stufe 1

Kara hat zwei Sensoren (Baum vorne und Kleeblatt unten). Je nachdem, was wahrgenommen wird, geht Kara einen Schritt oder hebt das Kleeblatt auf.

Es handelt sich also um Aufgaben der Stufe 1 der Komplexitätsklassifizierung aus Kapitel 5. Eine für die Schülerinnen und Schüler schwierigere Kara-Aufgabe ist eine, die mehrere Zustände neben dem Stoppzustand benötigt.

Beispiel 49:

Das gegebene Anfangsszenario ist in Abbildung 67 dargestellt. Kara soll von oben nach unten zwischen den Bäumen hin und her wandern und dabei das Muster der Kleeblätter invertieren, also ein Kleeblatt legen, wo das Feld frei ist, und es aufnehmen, wenn dort eins liegt.

Abbildung 67: Anfangsszenario

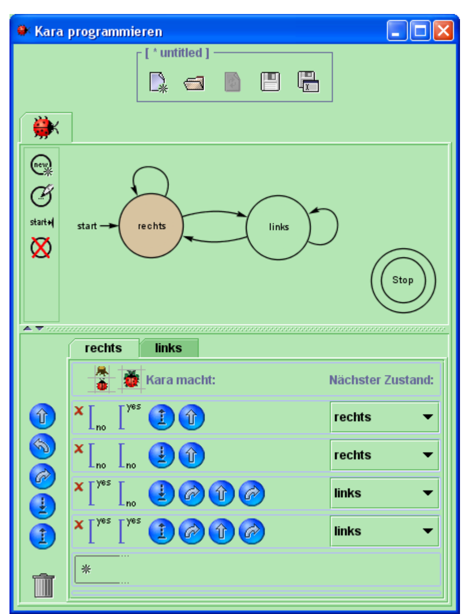


Abbildung 68: Kara-Programm der Stufe 2

Für die Programmierung sind mindestens zwei Zustände nötig, da zwischen dem Laufen nach rechts und nach links unterschieden werden muss. Analog zu dem Zustand „rechts“ sehen die Angaben im Zustand „links“ aus.
Diese Aufgabe gehört im Klassifikationsschema aus Kapitel 5 zur Stufe 2.

Die ersten Aufgaben, bei denen die Einführung weiterer Zustände unumgänglich war, haben vielen Schülerinnen und Schülern Probleme bereitet. Es musste eine Aufgabe aus diesem Gebiet gemeinsam gelöst und versucht werden, zu systematisieren, wann ein weiterer Zustand erforderlich ist, nämlich genau dann, wenn bestimmte Aspekte zu „merken/speichern“ sind. Erst anschließend konnten die Schülerinnen und Schüler selbstständig weiterarbeiten und eigenständige Transformationen finden.

Fazit:

Kara ist zwar kein reales technisches System, erfüllt sonst aber die Anforderungen der Kapitel 2 bis 4 und bietet eine Konfigurationsoberfläche mit endlichen Automaten – ein bedeutender Vorteil bei der späteren Systematisierung und ein anderer Ansatz, als ihn eine imperative Sprache bietet, wie in Kapitel 3 beschrieben. Kara sollte im Unterricht verwendet werden, aber nur zusätzlich zu einem realen System, um Gemeinsamkeiten erschließen zu können.

2. LEGO

Vorgehensweise:

In der zweiten Programmiereinheit wurde das reale technische Systeme LEGO verwendet (siehe Kapitel 3). Nach einer knappen Vorstellung der Lernumgebung haben die Schülerinnen und Schüler zunächst einfache Aufgaben bearbeitet. Als einfache LEGO-Aufgaben bezeichnet die Autorin solche, die aus einer Abfolge von Aktionen bestehen, wie man sie braucht, wenn man z. B. seinen

Anfangsbuchstaben abfahren will. Weiterhin sind einfache Aufgaben solche, die darauf beruhen, bei einer Unter-/Überschreitung eines Sensorwertes mit einer bestimmten Aktion zu reagieren. Wird z. B. der Berührsensor gedrückt, weil der Roboter gegen die Wand fährt, dann soll er rückwärts fahren oder drehen. Gemeint sind also Aufgaben der Komplexitätsstufe 1 nach Kapitel 5. Anschließend haben die Schülerinnen und Schüler eigene Ideen verwirklicht. Schwierige Aufgaben (der 2. Stufe) wurden in einem ersten Durchlauf nicht bearbeitet.

Reflexion:

Angemerkt sei an dieser Stelle, dass die technischen Probleme des Zusammenbaus von individuellen Fahrzeugen nicht unterschätzt werden dürfen, weil vielen Schülerinnen und Schülern die hierfür nötigen Kompetenzen bzgl. Mechanik und Statik fehlen. Bei der eigenen Wahl von Aufgaben, Ideen oder Projekten wurden die großen Unterschiede in der Komplexität sehr deutlich. Angefangen bei einer Schülergruppe, die lediglich eine Abfolge von Bewegungen programmiert hatte und bei der Präsentation ein passendes Lied abgespielt hat, so dass es aussah, als tanze der Roboter, bis hin zu komplexen Aufgaben mit geschachtelten Alternativen, geschachtelten Schleifen und nebenläufigen Prozessen war alles vorhanden. Ein komplexes eigenes Projekt einer Schülergruppe war z. B. eine Sortiermaschine, die „erkannt" hat, ob Legosteine auf dem Förderband liegen, und diese bis zu einem Lichtsensor transportiert hat. Der Lichtsensor hat dann „entschieden", ob es sich um einen weißen oder schwarzen Legostein handelt, und diese entsprechend auf einem Förderband nach links (weiße Steine) oder rechts (schwarze Steine) transportiert.

Fazit:

Das Lernziel, einfache Konfigurationen im LEGO-System selbstständig zu erstellen, haben alle Schülerinnen und Schüler erreicht. Die LEGO-RIS-Programmieroberfläche bietet vielfältige Möglichkeiten, die anfangs noch nicht relevant sind.

3. Systematisierung

Vorgehensweise 1:

Nach den konkreten Programmiererfahrungen folgten zwei Stunden, in denen die gemachten Erfahrungen systematisiert wurden. Die Begründung für dieses Vorgehen ergibt sich aus Überlegungen aus Kapitel 4. In diesen zwei Stunden wurde zunächst einmal versucht, die Erfahrungen mit Kara und den LEGO-Robotern zusammenzufügen.

Gemeinsam wurde folgende Tabelle herausgearbeitet:

	Eingaben/Abfragen	**(Re)aktionen**
Kara	Sensor Blatt unten Sensor Baum vorne / Baum rechts / Baum links Sensor Pilz vorne	Schritt vor links/rechts drehen um 90° Blatt legen Blatt aufnehmen
LEGO	Berührsensor (gedrückt / nicht gedrückt) Lichtsensor (schwarz / weiß)	Motor an Melodie spielen Zahlen ausgeben Lampe an

Tabelle 9: Ergebnis der Erarbeitungsphase: „Welche Gemeinsamkeiten gibt es zwischen LEGO und Kara?“

Reflexion 1:

Die Schülerinnen und Schüler haben trotz anfänglichem Zögern Gemeinsamkeiten erkannt, nämlich dass die Eingaben durch Sensoren und Sensorwerte erfolgen und dass es Reaktionen gibt, die der Roboter oder Kara ausführen können. Sie haben auch erkannt, dass bei bedingten Aktionen immer Sensorwerte auf die Über-/Unterschreitung bestimmter Schwellenwerte geprüft werden und in Schleifen ebenfalls die Frage nach Sensorwerten die Ausführung des Schleifenrumpfes bedingt, ohne jedoch bereits diese Fachbegriffe zu verwenden. *Das Sensor-Aktor-Prinzip ohne Variablen-Konzept (vgl. Kapitel 3) erlaubt es, dass bei Bedingungen stets nur Sensorwerte eine Rolle spielen und bei Ausgaben nur Aktoren angesprochen werden.* Diese didaktische Reduktion hat sich als sehr sinnvoll erwiesen.

Da die Systeme so ähnlich sind, aber eine vollkommen unterschiedliche Syntax haben (endliche Automaten bzw. imperative graphische Programmelemente), erschien es den Lernenden auch sinnvoll, eine Form zu nutzen, die in beiden Systemen gleichermaßen die vorhandenen Konfigurationen beschreibt und als Kommunikationsmittel zum Ideenaustausch zur Verfügung steht.

Vorgehensweise 2:

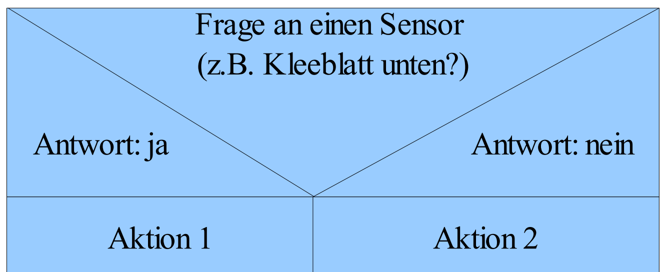
Es wurden Struktogramme als verbindendes Element eingeführt, wobei aufgrund des Sensor-Aktor-Prinzips einige Vereinfachungen vorgenommen werden konnten:

Wir vereinbarten Folgendes:

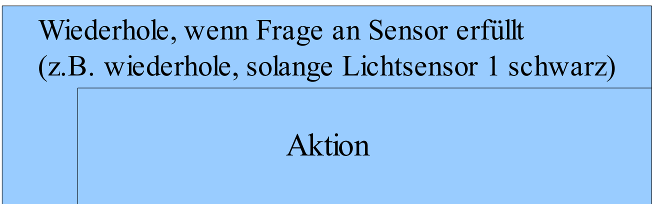
1. Aktionen (Motor an, Schritt vorwärts, Linksdrehung 90°, ...) stehen in Kästen.
2. Die nächste Aktion kommt in den Kasten darunter.

Aktion (z.B. Motor an)
Aktion (z.B. spiele Melodie 1)

3. Fragen werden immer an einen Sensor oder mehrere Sensoren gestellt.



4. Bei Wiederholungen fragt man einen Sensor oder mehrere Sensoren, ob die Aktionen wiederholt werden sollen.



Anschließend haben die Schülerinnen und Schüler herausgearbeitet, dass technische Alltagsgeräte Sensoren haben und damit dann Aktionen ausgelöst werden können. Später wurden Lichtsensoren in einem Strichcodescanner identifiziert und mit LEGO ein vereinfachter Strichcodescanner mit zwei Lichtsensoren rekonstruiert. Danach wurde überlegt, was es noch für Sensoren und Aktionen einer Maschine oder eines Gerätes geben könnte und welche Sensoren z. B. ein Wäschetrockner hat. Es wurde die Funktionsweise von Alltagsgeräten (Trockner, Blutdruckmessgerät, ...) mit Struktogrammen dargestellt.

Reflexion 2:

Die Syntax der Struktogramme lehnt sich an das LEGO-System an, so dass die Schülerinnen und Schüler keine Probleme damit hatten. Aber auch die Elemente von Kara konnten die Lernenden ganz schnell in diese Form transformieren. Allerdings erlaubt das LEGO-System nebenläufige Prozesse. Auch bei anderen technischen Geräten, die die Schülerinnen und Schüler aus ihrem Alltag

kennen, ist Nebenläufigkeit gegeben. Struktogramme erlauben diese Nebenläufigkeit nicht, was später zu „Fehlern“ geführt hat.
Beim Nachbau des Barcodescanners musste natürlich erst einmal die entsprechende Codierung erarbeitet werden. Aber den Schülerinnen und Schülern war vollkommen einsichtig, warum sie sich jetzt mit diesem Thema beschäftigen mussten. Dass Fachbegriffe aus anderen Gebieten erlernt werden mussten, um die Lösung zu gestalten, legitimierte für die Schülerinnen und Schüler den Aufwand.
Für die Darstellung der Funktionsweise anderer technischer Geräte gab es viele unterschiedliche Schülerergebnisse, z. B. zum Thema Trockner. Manche piepten lediglich bei trockener Wäsche. Andere haben nicht nur den Feuchtigkeitsgrad der Wäsche gemessen, sondern regelten die Heizung des Gebläses entsprechend den Einstellungen (Feinwäsche oder Baumwolle), was aber im Rahmen differenzierten Unterrichts gewollt ist.

Fazit:

Schülerinnen und Schüler erkennen nach Ansicht der Autorin den Sinn in Diagrammen, wenn sie damit ihre konkreten Erfahrungen beschreiben können. Es ist wichtig, unterschiedliche Programmierumgebungen zu betrachten, damit die Grundbausteine besser identifiziert werden können. Kara und LEGO eignen sich dafür in besonderer Weise, da sie optisch und im Ansatz ganz unterschiedlich aufgebaut sind, die elementaren Bausteine der Transformationen wie Alternative, Schleife usw. aber dieselben sind. Eine gemeinsame Notationsform, die unabhängig von der Programmierumgebung ist, erscheint den Lernenden sinnvoll. Der Nachteil der Struktogramme in punkto Nebenläufigkeit muss berücksichtigt werden. UML-Aktivitätsdiagramme sind gemäß den Ausführungen in Kapitel 3 vorzuziehen.
Die Schülerinnen und Schüler konnten technische Systeme beschreiben. Allerdings sind sie durch das LEGO-System, in dem die Sensoren und Aktoren begrenzt sind, beeinflusst. Ein weiteres System, in dem auch andere Sensoren (Temperatursensor, Feuchtesensor usw.) und andere Aktoren vorhanden sind, wäre daher sehr wünschenswert.

4. Karol

Vorgehensweise:

Um den Umgang mit Struktogrammen zu trainieren, wurde eine Einheit mit der Lernumgebung „Karol“ hinzugefügt. In dieser Lernumgebung werden die geschriebenen Programme automatisch in Struktogramme umgesetzt. Das Kommunikationsmittel der Karol-Lösungen waren daher in dieser Einheit Struktogramme. Auch konnten die Schülerinnen und Schüler vorgegebene

Struktogramme wieder in Code umsetzen und testen.

Reflexion:

Den Schülerinnen und Schülern unterliefen viele Syntaxfehler. Eine Hilfestellung des Lehrenden bei nicht funktionierenden Programmen wurde in dieser Einheit häufiger nachgefragt. Allerdings haben die Schülerinnen und Schüler den Umgang mit Struktogrammen trainiert.

Fazit:

Prinzipiell eignet sich diese Lernumgebung aufgrund der textbasierten Sprache nicht so sehr für die Anforderungen der vorhergehenden Kapitel. Es ist aber ein vertiefendes Üben von Struktogrammen oder UML-Diagrammen notwendig, weshalb diese Lernumgebung eingesetzt wurde. Geeignet wäre ein weiteres technisches Lernsystem, welches die Möglichkeit bietet, die Konfigurationen auch als UML-Aktivitätsdiagramm angeben zu können. Karol entfernt sich trotz Sensor-Aktor-Prinzip wieder von den Alltagsgeräten der Lebenswelt.

6.1.1.4 Fazit

In dieser Unterrichtssequenz haben die Schülerinnen und Schüler mit zwei sehr unterschiedlichen Programmierumgebungen (Kara und LEGO) gearbeitet und Erfahrungen gesammelt, die jedoch den didaktischen, inhaltlichen und methodischen Überlegungen aus dem ersten Teil dieser Arbeit weitgehend entsprechen. Aufgrund anschließender Vergleiche der beiden Systeme gelang es den Lernenden, die Grundbausteine ihrer Algorithmen zu identifizieren. Sie konnten erkennen, dass *unabhängig von der Syntax* dieser Sprachen bestimmte Konstrukte wie Alternative oder Schleife vorhanden sein müssen, die eine algorithmische Beschreibung von Problemlösungen ermöglichen. Struktogramme schienen den Schülerinnen und Schülern nach ihren Konfigurationserfahrungen sinnvoll, da ihre Lösungen damit übertragbar auf andere Systeme sind. Nach Meinung der Autorin wären den Lernenden Struktogramme weniger sinnvoll erschienen, wenn nur eine Lernumgebung Verwendung gefunden hätte, da man dann auch direkt diese Sprache hätte verwenden können. Für zwei Lernumgebungen mit ähnlicher Syntax gilt dasselbe.

6.1.1.5 Evaluation

Zu dem Zeitpunkt in der oben beschriebenen Unterrichtssequenz, als Automaten-Kara und das LEGO-System miteinander verglichen und gemeinsame Komponenten identifiziert wurden (es gibt Sensoren, es gibt Aktoren, man verknüpft sie für verschiedene Funktionen des Systems mit unterschiedlichen Bausteinen wie Alternativen, Sequenz usw.), also noch kein Transfer auf technische Systeme der Lebenswelt der Schülerinnen und Schüler stattgefunden hatte, wurde eine anonyme Befragung durchgeführt, um eventuell vorhandene Lernzuwächse bei den Lernenden erkennen zu können.

Zwei Lerngruppen wurden angehalten, ein bekanntes technisches System ihrer Lebenswelt zu beschreiben. Wie Frage lautete wahlweise: *„Wie funktioniert ein Wäschetrockner?“* bzw. *„Wie funktioniert ein Strichcodescanner im Supermarkt?“* Die Schülerinnen und Schüler sollten anonym und schriftlich in einer Vertretungsstunde aufschreiben, wie sie sich die Funktionsweise erklären, und zwar so detailliert wie möglich.

Diese Aufgabe wurde 26 Lernenden einer 7. Klasse gestellt, die *keinen* Informatikunterricht hatten, und dem oben genannten WPU-Informatikkurs Klasse 7 (25 Schülerinnen und Schüler, mit den oben beschriebenen Erfahrungen im Informatikunterricht). Es wird nochmals betont, dass bis zu dieser Aufgabenstellung noch keine technischen Systeme der Lebensumwelt beschrieben wurden, selbstredend auch nicht die o. g. Fragestellungen.

Die freien schriftlichen Antworten wurden anschließend folgendermaßen gruppiert:

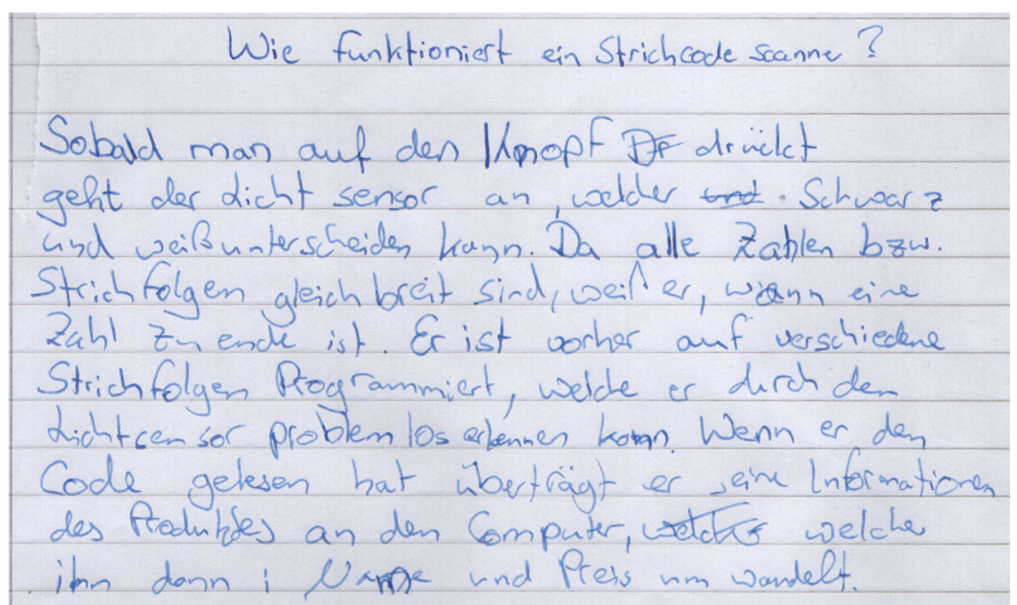
- In die Kategorie *„keine sinnvolle Angabe“*.
- Die Kategorie *„Bedienungsanleitung“* fasst Aussagen zusammen, die sich auf die *Bedienung* des technischen Geräts konzentrieren, aber nicht auf die vermutete Funktionsweise.
- Die Kategorie *„sichtbare Funktionalität“* fasst Aussagen zusammen, die detaillierter beschreiben, wie das System vermutlich arbeitet, aber nur Aspekte berücksichtigen, die von außen sichtbar sind, wie folgendes Beispiel 50 verdeutlicht.
- Die Kategorie *„technische Funktionalität“* fasst Aussagen zusammen, die detailliert die Funktionsweise des technischen Geräts beschreiben und dabei auch auf mögliche technische und algorithmische Anforderungen eingehen. (z. B. Identifikation von Sensoren, Teile der Konfiguration werden algorithmisch beschrieben usw.), wie Beispiel 51 zeigt.

Beispiel 50:

Abbildung 69: Schülerantwort auf die Frage „Wie funktioniert ein Wäschetrockner?“

„(1. Man stellt den Trockner an.) 2. Dann fängt der Trockner leicht an zu schleudern, durch die Löcher der Schleuder gelangt die trockene Luft an die Wäsche. Nach einer Zeit hört der Trockner auf zu schleudern, das liegt daran, dass der Trockner auf eine bestimmte Zeit eingestellt wurde.“

Beispiel 51:



Wie funktioniert ein Strichcode scanne?

Sobald man auf den Knopf drückt
geht der Licht sensor an, welcher Schwarz
und weiß unterscheiden kann. Da alle Zahlen bzw.
Strichfolgen gleich breit sind, weiß er, wann eine
Zahl zu ende ist. Er ist vorher auf verschiedene
Strichfolgen Programmiert, welche er durch den
Lichtcensor problemlos erkennen kann. Wenn er den
Code gelesen hat überträgt er seine Informationen
des Produktes an den Computer, welche
ihn dann in Name und Preis um wandelt.

Abbildung 70: Schülerantwort auf die Frage: „Wie funktioniert ein Strichcodescanner?“

„Sobald man auf den Knopf drückt, geht der Lichtsensor an, welcher schwarz und weiß unterscheiden kann. Da alle Zahlen bzw. Strichfolgen gleich breit sind, weiß er, wann eine Zahl zu Ende ist. Er ist vorher auf verschiedene

Strichfolgen programmiert, welche er durch den Lichtsensor problemlos erkennen kann. Wenn er den Code gelesen hat, überträgt er seine Informationen des Produkts an den Computer, welcher ihn dann in Name und Preis umwandelt.“

Die Ergebnisse dieser Befragung ergeben sich wie folgt:

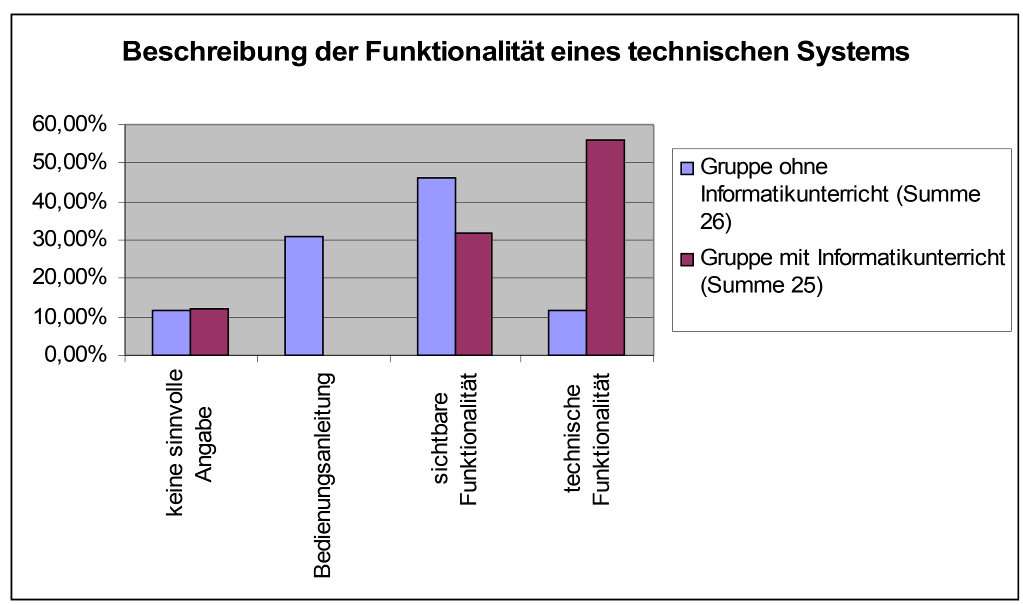


Abbildung 71: Häufigkeit der Klassen von Erläuterungen der Schülerinnen und Schüler zur Funktionsweise eines technischen Systems

Natürlich sind statistische Aussagen bei einer Gruppe von 25 bzw. 26 Schülerinnen und Schülern schwierig. Doch wenn man die Aussagen im einzelnen liest, sind die Erklärungen der Gruppe *mit* Kenntnissen in der Systemkonfiguration deutlich detaillierter. Man erkennt eine Verschiebung der Antworten. In der Gruppe ohne Programmierkenntnisse wird nur beschrieben, was von außen zu sehen ist. Die Gruppe mit der Erfahrung von Systemkonfigurationen hingegen denkt sich tiefer in die Systeme ein und stellt Vermutungen über Sensoren und Aktoren in diesen Systemen an. Die algorithmischen Zusammenhänge bei der Verarbeitung der Daten werden ebenfalls detaillierter betrachtet, auch wenn das von außen nicht sichtbar ist. Nach Ansicht der Autorin kann das wie folgt gedeutet werden: *Die gewonnenen Erkenntnisse über algorithmische Strukturen können nach dieser Unterrichtssequenz auch in bekannte Systeme der Lebenswelt, deren Funktionalität nicht Unterrichtsgegenstand war, transferiert werden.*

Damit wurde sich dem Ziel, nämlich Schülerinnen und Schüler ohne besondere informatische Begabung oder technisches Interesse zu befähigen, die sie umgebende technische Welt besser zu

durchdringen und die Funktionalität bekannter technischer Systeme algorithmisch beschreiben zu können, genähert.
Außerdem kann man vorsichtig Folgendes schließen: Die Gruppe *ohne* Informatikunterricht sieht sich sehr in der Rolle des Anwenders und Bedieners technischer Systeme, weshalb viele auch nur eine Bedienungsanweisung der Systeme gegeben haben. In der Gruppe *mit* Informatikunterricht haben sich die Häufigkeiten bestimmter Antworten verschoben. Keiner aus dieser Gruppe hat sich mit einer Bedienungsanleitung zufriedengegeben. Sie interessieren sich mehr für das „Innenleben“ der Systeme, sie sind auf dem Weg, sich als Konstrukteure von Systemkonfigurationen zu verstehen.
Die Ergebnisse sind deutlich genug, so dass es sich lohnen würde, dies genauer und umfangreicher zu untersuchen.

6.2 Programmieren in einer Lernumgebung und verschiedenen Oberflächen mit einer anschließenden Systematisierung

In diesem Abschnitt soll gezeigt werden, dass ein ebenfalls gangbarer Weg die Möglichkeit ist, mit ein und demselben System und verschiedenen Oberflächen die Grundbausteine von Algorithmen herauszuarbeiten.

Das PuMa-System bietet nach der Konzeption verschiedener Oberflächen, wie sie in Kapitel 5 vorgestellt wurde, die Möglichkeit, im selben System mit unterschiedlichen Beschreibungssprachen für Konfigurationen zu arbeiten. Dabei reichen die Beschreibungsmittel von Schaltwerken über Mealy-Maschinen zu UML-Aktivitätsdiagrammen und graphisch imperativen Sprachen.
Das PuMa-System kann mit verschiedenen Oberflächen die Möglichkeit bieten, dieselbe Problemstellung in *demselben System* mit verschiedenen „Sprachen“ zu lösen. Nehmen wir einmal an, die Aufgabe wäre folgende: Das Licht im Haus soll angehen, wenn es draußen dunkel wird. Die Schülerinnen und Schüler müssen also ihre Lampen an die Ausgänge stecken und einen Lichtsensor an einen Eingang. Dann wird das Potentiometer so eingestellt, dass hell und dunkel durch die beiden booleschen Werte dargestellt werden können. Jetzt konfigurieren die Lernenden dies System mit einem Schaltnetz, mit einem endlichen Automaten oder mit einem UML-Aktivitätsdiagramm. Die Verallgemeinerung hin zur Systematisierung erfolgt dann auf ähnlichem Wege wie in Kapitel 6.1. beschrieben. Die Schülerinnen und Schüler konfigurieren das PuMa-System in unterschiedlichen „Sprachen“, am besten sogar für dieselbe Aufgabe, und können so

über die Gemeinsamkeiten Algorithmenbausteine identifizieren. Darauf aufbauend kann dann ein weiterer Unterricht wie im Folgenden beschrieben aufbauen. Da dieselbe Problemstellung in unterschiedlichen Sprachen mit ihrer jeweils eigenen Syntax gelöst wird, lernen die Schülerinnen und Schüler ein Problem zu lösen und es in die Syntax der jeweiligen Sprache zu codieren. Sie lernen, Problemlösung und Codierung zu unterscheiden. Beim Übergang zu einer textbasierten Sprache vermischen sich Syntaxfehler dann weniger mit logischen Fehlern, weil den Schülerinnen und Schülern bewusst wird, dass die Problemlösung und die Codierung in die jeweilige Sprache zwei unterschiedliche Aufgaben sind. Zu diesem Aspekt wird eine zweite Unterrichtssequenz in diesem Kapitel näher aufgeführt.

6.3 Übergang zu textbasierten Sprachen am Beispiel „scratch“ und Delphi – eine Unterrichtssequenz im Schuljahr 2008/2009

Eine Übertragbarkeit und Systematisierung ist nur bei Kenntnis unterschiedlicher Beschreibungssprachen der Konfigurationen möglich, weil dann die Konzepte sinnvoll erscheinen und bei der Suche nach Gemeinsamkeiten zutage treten. Dies wurde in der ersten Unterrichtssequenz gezeigt. Sind die Schülerinnen und Schüler dem Ziel, zwischen Problemlösung und Codierung in vorgegebener Syntax zu unterscheiden, näher gekommen, ist auch der Schritt von graphischer zu textbasierter Codierung nicht mehr weit.

In Kapitel 5 wurde die Möglichkeit beschrieben, das PuMa-System auch mit dem „Picoboard“ und „scratch“ zu programmieren. Das fehlende Element, wirklich Aktoren anzusteuern, war in dieser Unterrichtssequenz nicht erforderlich. Eine Möglichkeit, dennoch Aktoren anzusteuern, wäre folgende Idee: Die Schülerinnen und Schüler erzeugen Bildschirmausgaben. Auf einem schwarzen Bildschirm werden durch einfache „scratch“-Befehle weiße Rechtecke erzeugt, die eine bestimmte Codierung darstellen. Vor dem Bildschirm ist eine Reihe Lichtsensoren angebracht, die diese weißen Rechtecke identifizieren und per digitaler Schaltung Aktoren ansteuern (siehe Abbildung 72 und 73).

Folgende Unterrichtseinheit in einem WPU-Informatikkurs Jahrgang 8 im Schuljahr 2008/2009 zeigt nun den Übergang zu Delphi:

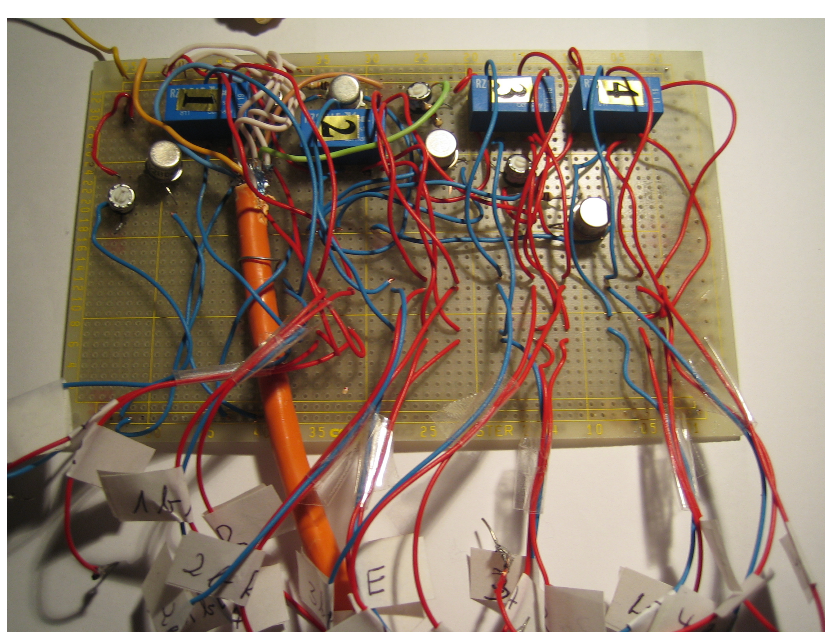
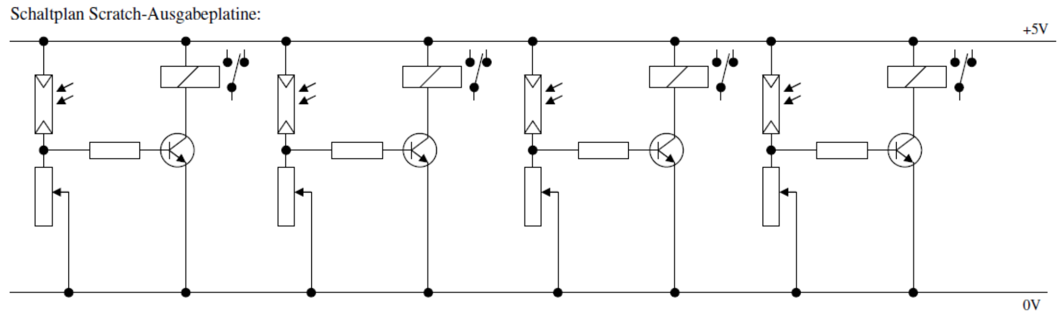


Abbildung 72: „scratch“-Ausgabealternative



4 x LDR 03
4 x Poti 10kOhm
4 x 1kOhm
4x NPN-Transistor
4x Doppelrelais

Abbildung 73: Schaltplan „scratch“-Ausgabealternative

Für den WPU-8-Kurs Informatik gelten dieselben Bedingungen in der Zusammensetzung der Schülerinnen und Schüler wie in der ersten Unterrichtssequenz. Es ist ein heterogener Kurs, in dem „Programmieren für Alle“ unterrichtet werden kann. Die Schülerinnen und Schüler sind mit der Konfiguration technischer Systeme vertraut und kennen die Programmierumgebung „scratch“ ohne technischen Zusammenhang.

Zeitbedarf im Unterricht für die Gesamteinheit PuMa:

Inhalte	**Zeitbedarf in Schulstunden**
Aufbauen der playmobil®-Häuser (entfällt im 2. Durchgang).	*2*
Einbinden und Ausprobieren des Picoboards, Entwicklung eigener Aufgabenstellungen, Präsentation.	1
Einbau von Sensoren im Haus und Anbindung an das Picoboard. Entwickeln eigener Aufgaben und Lösungen, Präsentation.	3
Physikalische Grundlagen bei der Schaltung von Sensoren, Anschließen von Sensoren und Aktoren am velleman-Board und Ausprobieren mit einer Demosoftware. (Keine Motoren.)	3
Von „scratch“ zu Delphi. Was fehlt in „scratch“? Wie ist „scratch“ systematisch aufgebaut? Entsprechungen in Delphi.	2
Realisierung eigener Ideen und Aufgaben mit Delphi und dem velleman-board. Inklusive Lösungen mechanischer Probleme mit den Motoren.	8

Tabelle 10: Übersicht der Unterrichtseinheit mit PuMa

Der Übergang von „scratch“ zu Delphi wird im Folgenden genauer erläutert:

Vorgehensweise:
Die Schülerinnen und Schüler kannten „scratch“ und haben sich den Umgang mit dem „Picoboard“ selbstständig erschlossen. Sie sollten eigene Aufgaben erfinden und ihre Programme präsentieren. Ein Ergebnis dieser Stunden ist z. B. das Programm eines Schülers mit mittleren Leistungen, bei dem eine Figur mit Hilfe der integrierten Sensoren gesteuert werden kann. Der Regler am „Picoboard“ verschiebt die Figur auf der Bühne entsprechend nach oben und unten bzw. nach links oder rechts. Die Richtung kann mit Hilfe des Lichtsensors ausgewählt werden. Bei einer Überschreitung bestimmter Werte, die der Geräuschsensor ausgibt, dreht sich die Figur, solange der Wert einen bestimmten Grenzwert übersteigt.
In der nachfolgenden Stunde sollten weitere PuMa-Sensoren angeschlossen werden. Die Schülerinnen und Schüler bauten z. B. eine Alarmanlage, die bei Auslösung des Berührsensors auf dem Bildschirm eine rote Lampe blinken ließ, u. a. Die Schülerinnen und Schüler machten sich so mit dem PuMa-System vertraut.
Nach diesen praktischen Übungen bekamen die Lernenden nacheinander drei Aufgaben. In der ersten Aufgabe sollten die „scratch“-Befehle systematisch in verschiedene Kategorien sortiert werden. Die Überschriften der Kategorien sollten selbst gewählt werden (die Lerngruppe kannte

bereits Struktogramme).

Ein Schülerergebnis, das von einem stärkeren Schüler präsentiert wurde, ergab folgendes Bild:

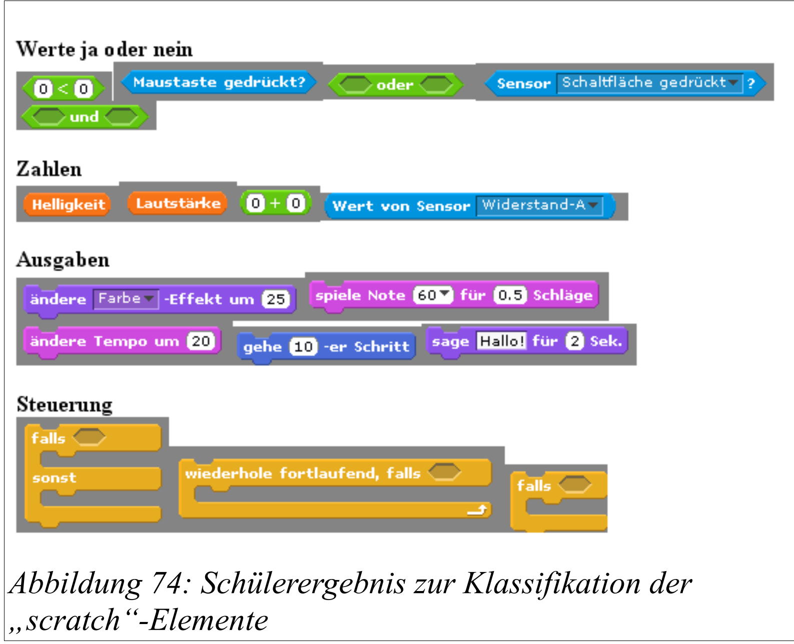
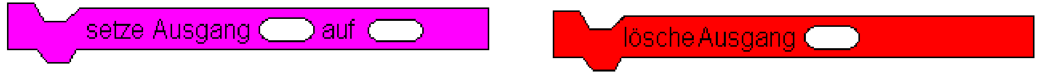


Abbildung 74: Schülerergebnis zur Klassifikation der „scratch"-Elemente

In der zweiten Aufgabe mussten die Schülerinnen und Schüler neue „scratch"-Befehle *erfinden*, die sie für ihr Puppenhausprojekt PuMa *mit Ausgabe* bräuchten, und in das Schema einordnen. Ein Schülerergebnis war folgende Lösung, die er auf Nachfrage in die oben genannte Kategorie „Ausgaben" einordnete:



In der dritten Aufgabe schließlich bekamen die Lernenden Bilddateien mit „scratch"-Befehlen und Bilddateien mit Delphi-Befehlen und mussten versuchen, diese entsprechend zuzuordnen.

Ein Auszug aus der Lösung zweier Schülerinnen mit mittleren Leistungen:

Werte ja oder nein

Sensor Schaltfläche gedrückt ? — und

```
(ReadDigitalChannel(1)=true)
```

```
((      ) and (      ))
```

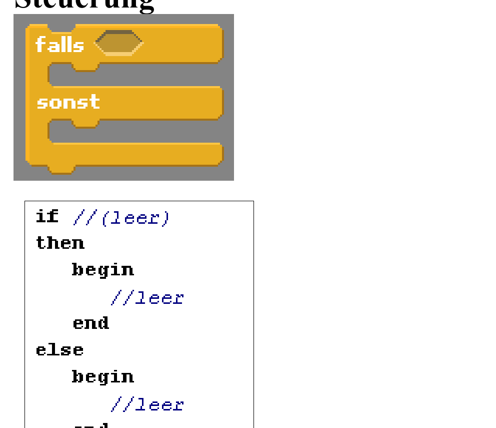
Zahlen

Wert von Sensor Widerstand-A

```
ReadDigitalChannel(1);
```

Ausgaben

Steuerung



Anschließend wurden mit den neuen Delphi-Konstrukten zunächst einfache Programme im PuMa-System realisiert, wie in Beispiel 52 beschrieben.

Beispiel 52:

- Lampe anschalten (mit dem Befehl SetDigitalChannel(1))
- Summer und Lampe anschalten
- später dann Aufgaben der 1. Kategorie (einfache Alternativen)

Reflexion:

Der Übergang von der graphischen Programmierumgebung „scratch“ zur textbasierten Programmierumgebung Delphi scheint gelungen zu sein. Den Schülerinnen und Schülern war es im Anschluss an diese Erarbeitungsphase möglich, ihre Projekte, die als Schülerprodukte in Kapitel 5 vorgestellt worden sind, in Delphi eigenständig umzusetzen. Nach Meinung der Autorin können die Schülerinnen und Schüler dieser Lerngruppe ganz offensichtlich zwischen einer Problemlösung und der Codierung in einer bestimmten Syntax trennen.

6.4 Zusammenfassung

Es wurde zwei Wege mit folgender Gemeinsamkeit gezeigt: Die Schülerinnen und Schüler beschäftigen sich mit Lernumgebungen, die den Anforderungen aus Kapitel 2 bis 4 bezüglich ihrer didaktischen Reduktion und ihrer inhaltlichen Ausrichtung genügen. Die Schülerinnen und Schüler lösen eigenständig Probleme, indem sie geeignete Konfigurationen finden. Egal, ob sie nun unterschiedliche Software in unterschiedlichen Systemen kennen lernen oder verschiedene Oberflächen in ein und demselben System. Wichtig ist, dass es sich um verschiedene Ansätze in der Programmierung handelt (funktional bzw. imperativ, mit endlichen Automaten oder Schaltnetzen, Schaltwerken oder auch mit Struktogrammen, UML-Aktivitätsdiagrammen oder graphisch-imperativen Sprachen). Nachdem die Schülerinnen und Schüler eigene Erfahrungen in den verschiedensten Systemen gewonnen haben, kann eine Systematisierung und Erarbeitung der Grundkomponenten einer Konfiguration bzw. eines Algorithmus erfolgreich sein. Wenn die Schülerinnen und Schüler erkennen, dass man unabhängig von den Eigenheiten der Sprachen Grundkonzepte herausarbeiten kann, dann können sie zwischen Problemlösung und Codierung unterscheiden. Außerdem können sie ihre Erkenntnisse auch auf unbekannte Systeme übertragen. Sie können Systeme ihrer Lebenswelt bzgl. der Ein- und Ausgaben analysieren und algorithmische Strukturen in der Konfiguration der Systeme erkennen.

Die Unterrichtserfahrung der Autorin zeigt, dass Aufgaben der 1. und 2. Stufe in den oben vorgestellten Kontexten in einem heterogenen Unterricht „für Alle“ in der Sekundarstufe I erfolgreich bearbeitet werden können.

7 Zusammenfassung der Arbeit

In den von der Gesellschaft für Informatik e.V. herausgegebenen „Bildungsstandards Informatik für die Sekundarstufe I“ (Puhlmann et al. 2008) wird indirekt eine Unterrichtseinheit „Programmieren“ für alle Schülerinnen und Schüler der Sekundarstufe I allgemeinbildender Schulen gefordert. Diese Arbeit zeigt ein Konzept, in dem diese Anforderung realisiert werden kann. Dabei liegt der Schwerpunkt in der Begründung und Darstellung dieses Konzeptes. Sich anschließende Arbeiten müssen später die konzeptionellen Entwürfe geeignet evaluieren.

Ein Unterricht im „Programmieren“ kann nur sinnvoll sein, wenn Kompetenzen vermittelt werden können, die in anderen Schulfächern oder Unterrichtseinheiten in dieser Form nicht vermittelbar sind und gleichzeitig einen allgemeinbildenden Wert darstellen, der den Lernenden hilft, die sie umgebende Lebenswelt besser zu verstehen und einordnen zu können. Das Ziel dieser Arbeit ist in diesem Zusammenhang, Schülerinnen und Schüler der Sekundarstufe I zu befähigen, die sie umgebende technische Welt besser durchdringen und unter anderem die Funktionsweise technischer Systeme algorithmisch beschreiben zu können. Dazu zeigt die vorliegende Arbeit einen inhaltlichen, methodischen und didaktischen Rahmen.

In Abgrenzung zu dem weit verbreiteten didaktischen Ansatz der objektorientierten Modellierung von Systemen setzt diese Arbeit einen Schwerpunkt auf die Konfiguration bekannter oder intuitiv erfassbarer Systeme. Dazu wird der Begriff der Systemkonfiguration geprägt, der die didaktische Reduktion der Tätigkeit des Programmierens in dieser Arbeit darstellt und wie folgt definiert wird:

> *Ist ein an sich bereits lauffähiges System mit Eingaben und Ausgaben gegeben, welches intuitiv zu erfassen oder ausreichend analysiert ist, dann bedeutet Systemkonfiguration, die Funktionalität dieses Systems so zu verändern, dass das System für eine bestimmte Aufgabe spezialisiert wird. Eine Systemkonfiguration ist dabei eine Funktion, die Eingaben auf Ausgaben abbildet. Als Beschreibungsmittel einer Konfiguration werden sogenannte Basiselemente verwendet, die ihrerseits Eingabewerte auf Ausgabewerte abbilden. Weiterhin gibt es Operationen auf diesen Basiselementen, die beliebig kombiniert werden können. Jede beliebige endliche Kombination von Anwendungen der Operationen und Basiselementen muss wieder eine gültige Konfiguration des Systems ergeben.*

Systemkonfigurationen können in ganz unterschiedlichen Beschreibungssprachen angegeben werden und erlauben damit vielfältige Zugänge für die Schülerinnen und Schüler. Damit können Lernende Konfigurationen nach dem Baukastenprinzip zusammensetzen und erhalten *immer* lauffähige Systeme, erzeugen also konkrete Produkte, und können so neben dem gewünschten analytischen Bottom-up-Entwurfsweg auch experimentell zu einer Lösung kommen. Das ist eine der Voraussetzungen dafür, dass ein *eigenständiger* Algorithmenentwurf für alle Schülerinnen und Schüler, auch für die leistungsschwächeren, gangbar ist.

Um die Stellung der Informatik als eins der wenigen technischen Fächer am Gymnasium zu betonen, beschränkt sich diese Arbeit auf die Konfiguration *technischer* Systeme. Die Lernenden beschreiben die Funktionsweise technischer Systeme ihrer Lebensumwelt algorithmisch, was dem Allgemeinbildungsanspruch eines Unterrichts „für Alle" genügt. Bestehen Konfigurationen allgemein aus einer Zusammensetzung der Algorithmenbausteine elementare Anweisung, Alternative, Sequenz und Schleife, so ergeben sich aus der Konfiguration *technischer* Systeme einige sinnvolle didaktische Aspekte:

- Konfigurationen technischer Systeme erlauben den Lernenden durch das Sensor-Aktor-Prinzip eine klare Trennung von Ein- und Ausgaben.
- Graphische Programmierumgebungen sind für die Konfiguration technischer Systeme ausreichend, was den Unterricht entlastet, da die Schülerinnen und Schüler nicht mit Syntaxproblemen konfrontiert werden.
- Technische Systeme lassen sich sogar im professionellen Bereich mit Schaltwerken oder Zustandsgraphen beschreiben, die die notwendigen Basisoperationen weiter eingeschränken. Bei imperativen Programmen ist z. B. die Verwendung von booleschen Variablen ausreichend.
- Im Falle der disjunkten Parallelität von Prozessen müssen die üblichen Algorithmenstrukturen erweitert werden, was dem Verständnis der Schülerinnen und Schüler aber entgegen kommt.
- Durch die Programmierung von realen Regel- und Steuerungssystemen ist die *Komplexität* der Aufgaben weiter beschränkt, wie unten angegeben wird.
- Konfigurationen technischer Systeme lassen sich mit ganz unterschiedlichen Beschreibungssprachen angeben, wie Mealy-Maschine, Schaltwerk, UML-Aktivitätsdiagramm, imperative und funktionale Sprachen.

Dabei zeigt diese Arbeit auch, wie die im bisherigen Unterricht verwendeten Struktogramme durch UML-Aktivitätsdiagramme abgelöst werden können.

Aktuelle Ergebnisse der Mathematikdidaktik werden verwendet, um zu zeigen, dass die *eigenständige* Suche nach Algorithmen oder Systemkonfigurationen bei den Schülerinnen und Schülern Kompetenzen fördert, die im bisherigen Unterricht unterrepräsentiert sind. Damit grenzt sich die vorliegende Arbeit auch gegen die didaktische Vorgehensweise der „Dekonstruktion von Systemen" ab. Da die Programmierung von Regel- und Steuerungssystemen die Komplexität der Aufgaben beschränkt, wird die eigenständige Algorithmensuche aller Lernenden erleichtert. Die Argumente des Dekonstruktionsansatzes für die Verwendung fertiger didaktischer Software, die von den Schülerinnen und Schülern nachvollzogen werden soll, liegen auch in der Realitätsnähe der damit zu behandelnden Probleme. Der Grundgedanke ist, dass die Programmierung von Systemen der Realität so komplex ist, dass sie von Lernenden nicht eigenständig entwickelt werden können. Durch die Beschränkung auf technische Systeme ist eine Behandlung genügend einfacher Probleme aber möglich, wie diese Arbeit zeigt. Der Realitätsbezug ergibt sich aus der Anwendung.
Um den Unterricht zweck- und produktorientiert zu gestalten, sieht die Autorin große Vorteile in *realen* Systemen, bei denen der Computer nur zur Darstellung der Konfiguration verwendet wird. Neben der Vorstellung des dazu geeigneten LEGO-RIS-Systems (LegoRIS 2009) beschreibt die Autorin ein selbst entwickeltes reales System, das PuMa-System. Ein Puppenhaus wird von kleinen Schülergruppen dabei mit Sensoren und Aktoren ausgestattet und geeignet konfiguriert. So dreht sich ein Ventilator, wenn es draußen zu heiß wird, oder die Markise wird auf Knopfdruck eingefahren. Das Puppenhaus-System kann auch mit einer Alarmanlage spezialisiert werden. Ist diese „scharf" geschaltet, ertönt ein Alarmsignal und eine rote LED blinkt beim Betreten der Treppe. Methodisch sollte sich die Arbeit mit dem PuMa-System am Projektunterricht orientieren.

Die Arbeit zeigt, welche Klassen von Aufgaben mit dem PuMa-System bearbeitet werden können und sieht in diesen eine sinnvolle Begrenzung für Aufgabentypen in der Sekundarstufe I.

> *Problemklasse 1:*
> *Aufgaben, deren Lösung als Schaltnetz, einfache Anweisungsfolge mit Alternativen ohne die Verwendung von Variablen oder als Mealy-Maschine mit einem Zustand angegeben werden kann.*

> *Problemklasse 2:*
> *Aufgaben, deren Lösung als Schaltwerk, Mealy-Maschine mit mehreren Zuständen oder Anweisungsfolge mit Verwendung von booleschen Variablen angegeben werden kann.*

Vergleicht man diese Abgrenzung z. B. mit den Themenschwerpunkten des Zentralabiturs in Niedersachsen (nibis 2009), dann werden Teile, die für die Oberstufe vorgesehen sind, z. B. Schaltwerke, in die Sekundarstufe I vorgezogen. Die Unterrichtserfahrung der Autorin zeigt, dass dieses möglich ist. Weitere Arbeiten sollten sich an dieser Stelle anschließen und diesen Bereich allgemeiner und vollständiger untersuchen.

Um die Funktionalität technischer Systeme der Lebenswelt, die nicht Unterrichtsgegenstand waren, algorithmisch beschreiben zu können, müssen die erarbeiteten Unterrichtsinhalte von den Schülerinnen und Schülern transferiert werden. Dazu müssen die Lernenden einerseits zwischen Problemlösung und Codierung unterscheiden können und andererseits die Grundideen von Algorithmen und Systemkonfigurationen verstanden haben. Diese Arbeit zeigt einen Weg, wie das erreicht werden kann. Wenn Schülerinnen und Schüler technische Systeme mit ganz unterschiedlichen Beschreibungsmitteln konfigurieren oder auch für ein und dasselbe System dieselbe Konfiguration in unterschiedlichen Darstellungsformen angeben, können sie anschließend Grundkonzepte identifizieren und systematisieren. Wenn zwischen Problemlösung und Codierung getrennt werden kann, gelingt auch der Übergang zu textbasierter Programmierung. Weitere Arbeiten sollten die in dieser Arbeit vorgestellten Ansätze näher untersuchen.

Mit der eigenständigen Suche einer Systemkonfiguration technischer realer Systeme wie der PuMa-Umgebung und der Beschränkung der Aufgabenkomplexität und Beschreibungsmittel ist ein Rahmen gegeben, mit dem ein Einstieg in die Programmierung in der Sekundarstufe I für alle Schülerinnen und Schüler erfolgreich gestaltet werden kann.

„Informatik für Alle – wie viel Programmierung braucht der Mensch?“ Das in dieser Arbeit vorgestellte und oben zusammengefasste Konzept ist für alle Schülerinnen und Schüler unabhängig von einem späteren Berufsweg geeignet, seine Ziele sind nach der Unterrichtserfahrung der Autorin erreichbar, und es bietet einen in sich geschlossenen Weg in die Programmierung.

Im Mittel ist der Anteil an Informatik in der Sekundarstufe I überall vergleichsweise gering (siehe dazu Hein 2006). Neben den anderen Themengebieten wie Datenbanken und Datenschutz oder Bildbearbeitung bleiben nicht viele Stunden für die Programmierung übrig. Eine eigenständige Systemkonfiguration technischer Systeme und der Transfer auf Alltagsgeräte füllt diese Zeit auch unter dem Anspruch der Allgemeinbildung sinnvoll aus.

Abbildungsverzeichnis

Abbildung 1: Schülermeinung zum Unterschied zwischen Automaten-Kara und Java........................8
Abbildung 2: Schülermeinung zum Unterschied zwischen Automaten-Kara und Java........................9
Abbildung 3: Aufgabe aus einem Schulbuch der Sekundarstufe I...14
Abbildung 4: Die fundamentale Idee „Algorithmisierung“...20
Abbildung 5: System aus Schaltern (Eingabe) und Lampen (Ausgabe) 21
Abbildung 6: Systemkonfiguration in einem System von Schaltern und Lampen............................22
Abbildung 7: Konfiguration eines Systems zur Jalousiesteuerung...25
Abbildung 8: Konfiguration eines Systems mittels Turingmaschine...25
Abbildung 9: Schema eines Techniksystems...33
Abbildung 10: Funktionsweise eines Blutdruckmessgerätes...35
Abbildung 11: Ein von Schülern mit dem LEGO-System konstruierter Strichcodescanner 38
Abbildung 12: Schülerprogramm des Strichcodescanners..39
Abbildung 13: Funktionalität eines Blutdruckmessgerätes...39
Abbildung 14: KOP-NOR-Verknüpfung 41
Abbildung 15: SR-Speicherglied im KOP..41
Abbildung 16: Schülerantwort als Struktogramm in einer Klausur...43
Abbildung 17: „Beweis“ in einem Mathematikschulbuch einer 8. Klasse...48
Abbildung 18: screenshot des Spiels "pong"...49
Abbildung 19: „scratch“-Programm „perfekte Zahl“...50
Abbildung 20: Java-Programm „perfekte Zahl“..50
Abbildung 22: Anfangsszenario...54
Abbildung 21: Kara-Programm..56
Abbildung 23: Lego Mindstorms RIS...59
Abbildung 24: Beispielprogramm eines Schülers einer 7. Klasse..60
Abbildung 25: Nebenläufige Prozesse...61
Abbildung 26: Funktionsweise eines Wäschetrockners...64
Abbildung 27: Beispiel eines UML-Aktivitätsdiagramms...66
Abbildung 28: Acht zentrale Aufgabentypen für nachhaltiges Lernen mit Beispielen 70
Abbildung 29: gelöste Beispielaufgabe..71
Abbildung 30: einfache Bestimmungsaufgabe...72
Abbildung 31: Beispiel einer offenen Aufgabe..73
Abbildung 32: Unterschiedliche Aufgabenarten im Mathematikunterricht..73
Abbildung 33: Verteilung der verschiedenen Aufgabenarten im Unterricht in Deutschland, den USA und Japan..74
Abbildung 34: Aussage einer Schülerin zur Fehlersuche...77

Abbildung 35: Beispiel einer Homematicsteuerung..84
Abbildung 36: Playmobil-Puppenhaus (Vorderansicht) 86
Abbildung 37: Rückseite des Playmobil-Hauses mit einigen Verkabelungen..................................86
Abbildung 38: Sensoren...86
Abbildung 39: Aktoren...86
Abbildung 40: velleman-Board..87
Abbildung 41: Eingangsplatine..88
Abbildung 42: Ausgangsplatine...88
Abbildung 43: Schülerprodukt 1..89
Abbildung 44: Schülerprodukt 3 in Arbeit...89
Abbildung 45: Schülerprodukt 3 nach Fertigstellung..89
Abbildung 46: Haustür von außen..90
Abbildung 47: Haustür von innen...90
Abbildung 48: graphische Darstellung der Antworten auf die Frage
„Was hat am meisten Spaß gemacht?“..92
Abbildung 49: Beurteilung ihres Lernzuwachses von Schülerinnen und Schülern mit
Interessenschwerpunkt in informatiknahen und informatikfremden Fächern...........93
Abbildung 50: Schülermeinung zum Unterschied LEGO und PuMa...96
Abbildung 51: Schülermeinung zum Unterschied LEGO und PuMa...96
Abbildung 52: Der Hardwaresimulator HASI von Eckart Modrow mit PuMa-Erweiterung............97
Abbildung 53: „Picoboard“...98
Abbildung 54: „scratch“-Programm mit „Picoboard“...99
Abbildung 55: Konfiguration als UML-Aktivitätsdiagramm..100
Abbildung 56: Konfiguration als Schaltwerk...101
Abbildung 57: Konfiguration als Mealy-Maschine...101
Abbildung 58: Konfiguration als graphische imperative Anweisungsfolge 101
Abbildung 59: Konfiguration der 1. Stufe...102
Abbildung 60: Konfiguration der 1. Stufe...103
Abbildung 61: Konfiguration der 1.Stufe..103
Abbildung 62: Konfiguration der 2. Stufe...104
Abbildung 63: Konfiguration der 2. Stufe...105
Abbildung 64: Konfiguration der 2. Stufe...105
Abbildung 65: Anfangsszenario...112
Abbildung 66: Kara-Programm der Stufe 1...112
Abbildung 67: Anfangsszenario...112
Abbildung 68: Kara-Programm der Stufe 2...113

Abbildung 69: Schülerantwort auf die Frage „Wie funktioniert ein Wäschetrockner?“..................120
Abbildung 70: Schülerantwort auf die Frage: „Wie funktioniert ein Strichcodescanner?“.............120
Abbildung 71: Häufigkeit der Klassen von Erläuterungen der Schülerinnen und Schüler zur Funktionsweise eines technischen Systems 121
Abbildung 72: „scratch“-Ausgabealternative...124
Abbildung 73: Schaltplan „scratch“-Ausgabealternative...124
Abbildung 74: Schülerergebnis zur Klassifikation der „scratch“-Elemente...................................126

Tabellenverzeichnis

Tabelle 1: Auszug aus den Themenschwerpunkten im Zentralabitur 2009 Informatik in Niedersachsen (nibis 2009)....12

Tabelle 2: Auszüge aus der inhaltlichen Struktur des Informatikunterrichts der Sekundarstufe I in Bayern (aus Weeger 2007)....13

Tabelle 3: Karas Sensoren und Aktionen....54

Tabelle 4: Einordnung der Aufgabe, eine Systemkonfiguration zu finden, gemäß Bruder....75

Tabelle 5: Einordnung der Strategie „Dekonstruktion von Systemen“ nach Bruder....75

Tabelle 6: Gegenüberstellung von Unterrichtskonzepten....83

Tabelle 7: Fragebogen zur Evaluation der Unterrichtseinheit PuMa....91

Tabelle 8: Übersicht Automaten-Kara, LEGO und Karol....111

Tabelle 9: Ergebnis der Erarbeitungsphase: „Welche Gemeinsamkeiten gibt es zwischen LEGO und Kara?“....115

Tabelle 10: Übersicht der Unterrichtseinheit mit PuMa....125

Literaturverzeichnis

Abelson, Sussman, Sussman 1996
Harold Abelson, Gerald Sussman, Julie Sussman: „Struktur und Interpretation von Computerprogrammen“, Springer-Verlag, 1996

Bartels 2009
Stefan Bartels: „Unterrichtskonzept Datenbanken“,
URL: http://www.vlin.de/vlin2/Modul2.html, Zugriff: 10.02.2009

Baumann 1992
Rüdeger Baumann: „Informatik für die Sekundarstufe II, Band 1“, Klett-Verlag, 1992

Baumann 1996
Rüdeger Baumann: „Didaktik der Informatik“, Klett-Verlag, 1996

Best 1995
Eike Best: „Semantik. Theorie sequentieller und paralleler Programmierung”, Vieweg-Verlag, 1995

Best, Strecker 2009
Eike Best, Kerstin Strecker: „Relational Semantics Revisited”, Proceedings of the Fifth Workshop on Structural Operational Semantics (SOS 2008) ‚Elsevier-Verlag, 2009
URL: http://dx.doi.org/10.1016/j.entcs.2009.07.072, Zugriff: 21.08.2009

Bildungsstandards 2003
Beschlüsse der Kultusministerkonferenz: „Bildungsstandards im Fach Mathematik für den Mittleren Schulabschluss“, Beschluss vom 04.12.2003
URL:http://www.kmk.org/fileadmin/veroeffentlichungen_beschluesse/2003/2003_12_04-Bildungsstandards-Mathe-Mittleren-SA.pdf, Zugriff: 10.07.2009

Bönsch 2004
Manfred Bönsch: „Differenzierung in Schule und Unterricht“, Oldenbourg-Verlag, 2004

Börstler 2007
Jürgen Börstler: „Objektorientiertes Programmieren – Machen wir irgendwas falsch?“, in Sigrid Schubert (Hrsg.): „Didaktik der Informatik in Theorie und Praxis“, INFOS 2007, 12. GI-Fachtagung Informatik und Schule, 19. – 21. September 2007 in Siegen, GI-Edition Lecture Notes

Boulay et al. 1999
Benedict Du Boulay, Tim O´Shea, John Monk: „The black box inside the glass box: Presenting computing concepts to novices.” International Journal of Human-Computer Studies, 1999

Brinda 2004
Torsten Brinda: „Didaktisches System für objektorientiertes Modellieren im Informatik-unterricht der Sekundarstufe II“, Dissertation, Fachbereich 12 – Elektrotechnik und Informatik der Universität Siegen, 2004

Bruder, Leuders, Büchter 2008
Regina Bruder, Timo Leuders, Andreas Büchter: „Mathematikunterricht entwickeln. Bausteine für kompetenzorientiertes Unterrichten“, Cornelsen-Verlag, 2008

Bruner 1976
Jerome S. Bruner: „Der Prozess der Erziehung“, Berlin-Verlag, Düsseldorf, 1976

Brusilovsky et al. 1997
Peter Brusilovsky, Eduardo Calabrese, Jozef Hvorecky, Anatoly Kouchnirenko, Philip Miller: „Mini-languages: A Way to Learn Programming Principles“, Education and Information Technologies 2(1), 1997

Büdding 2007
Hendrik Büdding: „Einführung in visuelle Programmiersprachen und Mobile Endgeräte“ in Sigrid Schubert (Hrsg.): „Didaktik der Informatik in Theorie und Praxis“, INFOS 2007, 12. GI-Fachtagung Informatik und Schule, 19. – 21. September 2007 in Siegen, GI-Edition Lecture Notes

conrad 2009
URL: http://www.conrad.de, Zugriff: 10.07.2009

contronics 2009
URL: http://www.contronics.de/html/homematic.html, Zugriff: 15.02.2009

Diethelm 2007
Ira Diethelm: „strictly models and objects first – Unterrichtskonzept und -methodik für objektorientierte Modellierung im Informatikunterricht“, Dissertation, Universität Kassel, 2007

Engelmann 2007
Lutz Engelmann (Hrsg.): „Duden Informatik. Lehrbuch S2“, Duden Paetec Schulbuchverlag, 2007

Engelmann 2008
Lutz Engelmann (Hrsg.): „Duden Informatik S1. Informatische Grundbildung“, Duden Paetec Schulbuchverlag, 2008

EPA 2004
URL:http://www.kmk.org/fileadmin/veroeffentlichungen_beschluesse/1989/1989_12_01_EPA_Informatik.pdf, Zugriff: 06.07.2009

Floyd, Züllighoven 2002
Christiane Floyd, Heinz Züllighoven: „Softwaretechnik“, in: Peter Rechenberg, Gustav Pomberger (Hrsg.) „Informatik-Handbuch“, Hanser-Verlag, 2002

Frey, Hubwieser, Winhard 2004
Elke Frey, Peter Hubwieser, Ferdinand Winhard: „Informatik 1. Objekte, Strukturen, Algortithmen“, Klett-Verlag, 2004

Griesel 1992
Heinz Griesel, Helmut Postel (Hrsg.): „Informatik heute. Algorithmen und Datenstrukturen Band 1“, Schroedel-Schulbuchverlag, 1992

Hampel, Magenheim, Schulte 1999
Thorsten Hampel, Johannes Magenheim, Carsten Schulte: „Dekonstruktion von Informatiksystemen als Unterrichtsmethode – Zugang zu objektorientierten Sichtweisen im Informatikunterricht“, in Andreas Schwill (Hrsg.): „Informatik und Schule. Fachspezifische und fachübergreifende didaktische Konzepte“, INFOS 1999, 8. GI-Fachtagung Informatik und Schule, Springer Verlag, 1999

Hartmann, Kussmann, Scherweit 2008
Elke Hartmann, Michael Kussmann, Steffen Scherweit (Hrsg.): „Technik und Bildung in Deutschland. Technik in den Lehrplänen allgemeinbildender Schulen. Eine Dokumentation und Analyse“, VDI-Report 38, 2008

Hein 2006
Christian Hein: „Technikunterricht, Informationstechnik und bildungspolitische Probleme“; in: LOG IN, 26. Jg., 2006, Heft 143

Herget, Jahnke, Kroll 2008
Wilfried Herget, Thomas Jahnke, Wolfgang Kroll: „Produktive Aufgaben für den Mathematikunterricht in der Sekundarstufe I“, Cornelsen-Verlag, 2008

Hermes 1990
Alfred Hermes, Dieter Stobbe: „Informatik Eins“, Klett-Verlag, 1990

Heymann 1996
Hans Werner Heymann: „Allgemeinbildung und Mathematik“, Beltz Verlag, 1996

homematic 2009
URL: http://www.homematic.com, Zugriff: 15.02.2009

Hubwieser 2000
Peter Hubwieser: „Informatik am Gymnasium. Ein Gesamtkonzept für einen zeitgemäßen Informatikunterricht“, Habilitationsschrift, Fakultät für Informatik, Technische Universität München, 2000

Hubwieser 2004
Peter Hubwieser: „Didaktik der Informatik. Grundlagen, Konzepte, Beispiele“, Springer-Verlag, 2004

Karol 2009
URL: http://www.schule.bayern.de/karol, Zugriff: 14.07.2009

Klaeren, Sperber 2007
Herbert Klaeren, Michael Sperber: „Die Macht der Abstraktion. Einführung in die Programmierung“, Teubner-Verlag, 2007

Lambacher Schweizer 2006
Dieter Brandt, Günther Dopfer, Rolf Reimer: „Lambacher Schweizer. Mathematik für Gymnasien 7“ Ausgabe Niedersachsen, Klett-Verlag, 2006

Lambacher Schweizer 2007
Manfred Baum, Detlef Dornieden, Heiko Harborth, Ulrich Schönbach: „Lambacher Schweizer. Mathematik für Gymnasien 8“, Klett-Verlag, 2007

LegoRIS 2009
URL: http://shop.lego.com/Product/?p=3804, Zugriff: 10.07.2009

Meyer 1990
Meyers Taschenlexikon, Meyers Lexikonverlag, Mannheim, 1990

Meyer 2007
URL: http://lexikon.meyers.de/index.php?title=Technik&oldid=281222,
Zugriff: 23.05.2008

MID 2009
URL:http://www.mid.de/fileadmin/documents/pdf/produktinformationen/Innovator_2008_Datenblaetter.pdf, Zugriff: 06.07.2009

Modrow 1991
Eckart Modrow: „Zur Didaktik des Informatik-Unterrichts. Band 1“, Dümmler Verlag, 1991

Modrow 1992
Eckart Modrow: „Zur Didaktik des Informatik-Unterrichts. Band 2“, Dümmler Verlag, 1992

Modrow 2004
Eckart Modrow: „Technische Informatik mit Delphi“, emu-online Verlag, 2004

Neubrand 2002
Johanna Neubrand: „Eine Klassifikation mathematischer Aufgaben zur Analyse von Unterrichtssituationen“, Verlag Franzbecker, 2002

nibis 2009
URL:http://www.nibis.de/nli1/gohrgs/zentralabitur/zentralabitur_2009/18Informatik2009.pdf, Zugriff: 06.07.09

Oesterreich 2006
Bernd Oesterreich: „Analyse und Design mit UML 2.1“, Oldenbourg-Verlag, 2006

Oose 2001
Oose GmbH: „Objekorientierte Softwareentwicklung: Analyse und Design mit der UML“, Seminarunterlagen, 2001, URL der GmbH: http://www.oose.de/,
Zugriff: 08.08.2009

Pepper, Hofstedt 2006
Peter Pepper, Petra Hofstedt: „Funktionale Programmierung“, Springer-Verlag, 2006

Peterßen 1999
Wilhelm Peterßen: „Kleines Methoden-Lexikon“, Oldenbourg-Verlag, 1999

Picoboard 2009
URL: http://www.picocricket.com/picoboard.html, Zugriff: 15.02.2009

Pomberger, Rechenberg 2002
Peter Rechenberg, Gustav Pomberger (Hrsg.): „Informatik-Handbuch“, Hanser-Verlag, 2002

Poswig 1996
Jörg Poswig: „Visuelle Programmierung“, Hanser-Verlag, 1996

Prante 1978
Manfred Prante, Wolfgang Tofahrn: „Informatik“, Schöningh-Verlag, 1978

Puhlmann et al. 2008
Gesellschaft für Informatik e.V.: „Grundsätze und Standards für die Informatik in der Schule. Bildungsstandards Informatik für die Sekundarstufe I“, Beilage zu LOG IN, 28. Jg., 2008, Heft Nr. 150/151

Rabel, Oldenburg 2009
Magnus Rabel, Reinhard Oldenburg: „Erwartungen und Wertungen von Schülern und Studenten“, INFOS 2009, GI-Fachtagung Informatik und Schule, September 2009, *Entwurfspapier*

Rauber, Rünger 2007
Thomas Rauber, Gudula Rünger: „Parallele Programmierung“, Springer-Verlag, 2007

Reichert, Nievergelt, Hartmann 2005
Raimond Reichert, Jürg Nievergelt, Werner Hartmann: „Programmieren mit Kara“, Springer-Verlag, 2005

Schiffmann, Schmitz 2003
Wolfram Schiffmann, Robert Schmitz: „Technische Informatik 1“, Springer-Verlag, 2003

Schmuller 2000
Joseph Schmuller: „Jetzt lerne ich UML“, Markt+Technik Verlag, 2000

Schöning 1992
Uwe Schöning: „Theoretische Informatik kurz gefasst“, BI-Wissenschaftsverlag, 1992

Schubert, Schwill 2004
Sigrid Schubert, Andreas Schwill: „Didaktik der Informatik“, Spektrum Verlag, 2004

Schwill 1993
Andreas Schwill: „Fundamentale Ideen der Informatik“, Zentralblatt für Didaktik der Mathematik 1:20 – 31, 1993

scratch 2009a
URL: http://scratch.mit.edu/, Zugriff: 15.02.2009

scratch 2009b
URL: http://web.media.mit.edu/~mres/papers/sigcse-08.pdf, Zugriff: 15.02.2009

SPS-Lehrgang 2009
URL: http://www.sps-lehrgang.de/kontaktplan-kop/, Zugriff: 09.07.2009

vellemann 2009
URL: http://www.velleman.be/ot/de/product/view/?id=351346, Zugriff: 14.07.2009

Wagenknecht 2004
Christian Wagenknecht: „Programmierparadigmen. Eine Einführung auf der Grundlage von Scheme“, Teubner Verlag, 2004

Weeger 2007
Moritz Weeger: „Synopse zum Informatikunterricht in Deutschland. Analyse der informatischen Bildung der allgemein bildenden Schulen – durchgeführt auf der Basis existierender Lehrpläne und Richtlinien“, Bakkalaureatsarbeit, Technische Universität Dresden, Arbeitsgruppe Didaktik der Informatik, 2007,
URL: http://dil.inf.tu-dresden.de/schule/Weeger/output.inf.tu-dresden.de/homepages/uploads/media/synopse_weeger.pdf, Zugriff: 15.02.2009

Wiesner, Brinda 2007
Bernhard Wiesner, Torsten Brinda: „Erfahrungen bei der Vermittlung algorithmischer Grundstrukturen im Informatikunterricht der Realschule mit einem Robotersystem“, in Sigrid Schubert (Hrsg.): „Didaktik der Informatik in Theorie und Praxis“, INFOS 2007, 12. GI-Fachtagung Informatik und Schule, 19. – 21. September 2007 in Siegen, GI-Edition Lecture Notes

wikipedia 2008a
URL: http://de.wikipedia.org/wiki/Steuerungstechnik, Zugriff: 17.10.2008

wikipedia 2008b
URL:http://de.wikipedia.org/wiki/Regelungstechnik#Technische_Anwendungen, Zugriff: 17.10.2008

wikipedia 2008c
URL: http://de.wikipedia.org/wiki/Speicherprogrammierbare_Steuerung, Zugriff: 17.10.2008

wikipedia 2009d
URL: http://de.wikipedia.org/wiki/Gymnasium Zugriff: 14.02.2009

wikipedia 2009e
URL: http://de.wikipedia.org/wiki/Nassi-Shneiderman-Diagramm, Zugriff: 20.07.2009

Witten 2003
Helmut Witten: „Allgemeinbildender Informatikunterricht? Ein neuer Blick auf H.W. Heymanns Aufgaben allgemeinbildender Schulen“, in: Peter Hubwieser (Hrsg.): „Informatische Fachkonzepte im Unterricht“, INFOS 2003, 10. GI-Fachtagung Informatik und Schule, GI-Lecture Notes

Lebenslauf

Name:	Kerstin Maike Strecker, geb. Richter
Geburtstag:	06. Mai 1973 in Kiel
Familienstand:	Verheiratet mit dem Landwirt Horst Strecker, zwei Kinder
	Lukas Lennart Strecker (*1999) Janne Rubi Strecker (*2001)
Schulbildung:	1979 – 1983 Grundschule Nörten-Hardenberg und Katlenburg
	1983 – 1985 Orientierungsstufe Katlenburg-Lindau
	1985 – 1992 Gymnasium Corvinianum Northeim
	1992 Abitur
Studium und Abschlüsse:	1992 – 1998 Studium der Informatik mit Anwendungsfach Medizinische Informatik an der Universität Hildesheim
	1995 Vordiplom in Informatik, Note: sehr gut
	1998 Diplom in Informatik, Note: sehr gut
	2003 – 2004 Studium der Schulpädagogik und Didaktik an der Georg-August-Universität Göttingen
	2004 Master of Arts in Schulpädagogik und Didaktik, Note: sehr gut
	2007 Zweite Staatsprüfung für das Lehramt an Gymnasien mit den Fächern Informatik und Mathematik, Note: sehr gut
Berufliche Laufbahn:	1998 – 2000 Informatikerin bei der Firma ORGAPLAN in Northeim
	2000 – 2002 Informatikerin bei der Firma GreCon in Alfeld/Leine
	2002 – 2005 Wissenschaftliche Mitarbeiterin am Institut für Informatik der Georg-August-Universität Göttingen
	2005 – 2007 Studienreferendarin an der Geschwister-Scholl-Gesamtschule in Göttingen
	2007 – heute Studienrätin am Max-Planck-Gymnasium Göttingen, Fächer: Informatik und Mathematik