3 Entwurf einer Unterrichtsmethodik

Nachdem wir im ersten Kapitel geklärt haben, was wir unter Informatikunterricht verstehen wollen und im zweiten dargelegt wurde, warum dieser unserer Meinung nach eine unverzichtbare Komponente einer tragfähigen Allgemeinbildung darstellt, soll in diesem Kapitel nun eine geeignete Unterrichtsmethodik skizziert werden. Dies geschieht bewusst *vor* der Auswahl möglicher Lerninhalte, da die Beurteilung der Umsetzbarkeit dieser Inhalte stark von der geplanten Methodik abhängt. Überhaupt ist eine Abgrenzung zwischen dem *Wie* (der Methodik) und dem *Was* (den Lerninhalten) des Informatikunterrichts oft sehr schwierig. So stellt Modellierung und Simulation einerseits ein wichtiges methodisches Grundprinzip des Unterrichts dar, während andererseits gewisse Modellierungstechniken als zentrale Lerninhalte auftreten.

3.1 Lernpsychologisches Fundament

Aufbauend auf den Erkenntnissen der Lernpsychologie (siehe Teil A, Kapitel 1) sollte der Unterricht die folgenden Forderungen erfüllen:

- Erzeugung einer entspannten Arbeitsatmosphäre, in der die Schüler ohne Stoffdruck Zeit haben, auch spezielle Interessen und Bedürfnisse zu befriedigen. Damit sollen Motivation und Aufmerksamkeit der Schüler gefördert und dauerhaft aufrechterhalten werden (vgl. Teil A, Abschnitt 1.1.1).
- Einordnung der Lerninhalte in größere Sinnzusammenhänge sowie eine deutliche Strukturierung der Stoffe, um den Schülern die Bildung *präpositionaler Netzwerke* zu ermöglichen (vgl. Teil A, Abschnitt 1.1.2).
- Förderung einer aktiven Auseinandersetzung mit dem Stoff, wobei unbedingt vor der Präsentation von Lösungen ein ausreichendes Problembewusstsein erzeugt werden muss. Diese Forderungen entstammen den gemeinsamen Erkenntnissen aller gemäßigt konstruktivistischen Lernansätze, die betonen, dass Wissen vom Lernenden aktiv konstruiert wird (vgl. Teil A, Abschnitt 1.4).
- Anbieten verschiedener Perspektiven und Zugänge zum selben Thema im Sinne von *Cognitive Flexibility* (vgl. Teil A, Abschnitt 1.4).
- Erzeugung möglichst authentischer Problemsituationen, um mit den Schülern Problemlöseverhalten in einer Umgebung zu trainieren, in der dieses Verhalten tatsächlich auch benötigt wird (*Cognitive Apprenticeship*, vgl. Teil A, Abschnitt 1.4).

- Altersgemäße Darbietung der Lerninhalte. Nach den Erkenntnissen der Entwicklungspsychologie Piagets (vgl. Teil A, Abschnitt 1.3) entwickelt sich z.B. in der Gegend der 6. Jahrgangsstufe bei den Kindern die Fähigkeit, hypothetische Vorgänge in ihre Überlegungen mit einzubeziehen, die für die Erarbeitung von Problemlösestrategien im Rahmen des Informatikunterrichts (siehe nächster Abschnitt) von großer Bedeutung ist.

3.2 Methodische Prinzipien

Neben den allgemeinen didaktischen Prinzipien (siehe Teil A, Kapitel 2) gibt es weitere methodische Prinzipien, die speziell für den Informatikunterricht von besonderer Bedeutung sind, wie Problemorientierung oder Modellbildung und Simulation.

3.2.1 Problemorientierung

Nach den Erfahrungen mit der Umsetzung abstrakter Konzepte wie der Gruppentheorie oder der Abbildungsgeometrie im Mathematikunterricht des Gymnasiums scheint eine Vermittlung der naturgemäß abstrakten informatischen Lerninhalte nur dann erfolgversprechend, wenn durch konkrete, anschauliche Problemstellungen eine erhöhte Aufnahmebereitschaft der Schüler geschaffen wird.

Edelmann (1986) beschreibt den Begriff „Problem“ (aus lernpsychologischer Sicht) folgendermaßen:

> Ein Problem ist also durch drei Komponenten gekennzeichnet:
> – Unerwünschter Anfangszustand,
> – erwünschter Zielzustand,
> – Barriere, die die Überführung des Anfangszustandes in den Zielzustand im Augenblick verhindert.

Davon zu unterscheiden ist die *Aufgabe* (ebenfalls nach Edelmann (1986)):

> Bei einer Aufgabe verfügen wir über Regeln (Wissen, Know-how), wie die Lösung zu erreichen ist.

Die Informatik kann in Form von Strukturierungshilfen und Simulationsmöglichkeiten besonders wertvolle Beiträge zur Problemlösefähigkeit der Schüler leisten, wie Brauer u. Brauer (1973) bereits zur Zeit der „Rechnerkunde“ feststellten:

> Also sollte das Hauptgewicht nicht gelegt werden auf das Kennenlernen des technischen Aufbaus von Rechenanlagen oder auf das Erlernen einer Programmiersprache. Die Schüler sollten vielmehr eingeführt werden in
>
> 1. Methoden der Strukturierung, Mathematisierung und Algorithmisierung von Problemkreisen aus verschiedensten Gebieten <d.h. Methoden der Modellbildung und Problemlösung> sowie

2. vor allem die Methoden des systematischen Programmierens und
3. Möglichkeiten des Einsatzes von Datenverarbeitungssystemen zur Behandlung komplexer Aufgaben (lnformationssysteme, heuristische Programmierung, Simulation).

Mittlerweile ist das *Problemlösen mit Informatiksystemen* eine der allgemein anerkannten Leitlinien der informatischen Bildung (siehe Friedrich (1995)). Die Struktur des Problemlöseprozesses stellt deshalb gerade für den Informatikunterricht ganz im Sinne von Roth (1963) (vgl. Teil A, Abschnitt 4.5.1) ein geeignetes Mittel zur *Artikulation* des Informatikunterrichts dar. Jeder neue Stoff sollte anhand von Problemen aus dem Erfahrungsbereich der Schüler (s.a. Robinsohn (1971)) eingeführt werden: Einer Motivationsstufe folgt eine Stufe der Schwierigkeiten, die dann in einer nächsten Stufe gelöst werden.

Die didaktischen Fähigkeiten des Lehrers zeigen sich dabei in der Auswahl geeigneter Probleme, deren Komplexität einerseits so hoch sein sollte, dass sie von den Schülern ohne die zu erlernenden Konzepte nicht oder nur unter erheblich höherem Aufwand gelöst werden können. Andererseits darf der intellektuelle Horizont der Schüler nicht überschritten werden. Im Sinne von Edelmann (1986) wären Probleme, die sich durch *Anwendung von Strategien* oder durch *Systemdenken* lösen lassen, optimal. Das Ergebnis des Problemlöseprozesses ist dabei im Vergleich zu konventionellen Unterrichtsfächern aus dem mathematisch-naturwissenschaftlichen Bereich meist wesentlich offener.

Eine strikte Problemorientierung kann den Informatikunterricht auch davor bewahren, in die Niederungen reiner Produktschulung abzufallen (siehe auch Abschnitt 1.1.2), wo oft das konkrete System zum Ausgangspunkt unterrichtlichen Handelns gemacht wird. Typischerweise führt eine solche Werkzeugorientierung, wie sie oft beim „Programmierunterricht“ vergangener Tage zu finden war, bei den Schülern zur Frage: „Jetzt habe ich ein schönes Werkzeug kennen gelernt, was soll ich nun damit anfangen?“

3.2.2 Modellbildung und Simulation

Im Gegensatz zu anderen Konzeptionen (Fakultätentag (1996), Hoppe u. Luther (1996), Rechenberg (1997)) betrachten wir den Prozess von Modellbildung und Simulation nicht als *Lerninhalt*, sondern als durchgängiges *Prinzip* der Unterrichtsgestaltung. Deshalb findet man diese Schlagwortkombination auch nicht unter den später vorgeschlagenen Lerninhalten. Unser Informatikunterricht beschäftigt sich nicht nur mit Modellbildung und Simulation, unser Unterricht besteht im Wesentlichen aus Modellbildung und Simulation, wie wir in Abschnitt 3.3.2 ausführlich darlegen werden. Spezielle, schülergemäße *Modellierungstechniken* gehören dagegen durchaus zu den vorgeschlagenen Lerninhalten.

3.3 Organisationsrahmen für den Informatikunterricht

Trotz aller Verschiedenheiten bezüglich der Situation an verschiedenen Schularten und in den einzelnen Bundesländer gibt es einige Rahmenbedingungen, deren Einhaltung für alle Arten von Informatikunterricht anzuraten ist.

3.3.1 Verankerung im Pflichtfachbereich

Unter Berücksichtigung der gegenwärtigen Schulsituation ist die Vermittlung der entscheidenden allgemein bildenden Lerninhalte aus dem Bereich der Informatik nur innerhalb eines eigenen, fest installierten Pflichtfaches möglich. Dafür kann man zumindest drei Begründungen anführen:

Lehrerausbildung. Nur im Rahmen eines speziellen Pflichtfaches „Informatik" kann sichergestellt werden, dass der Unterricht von Lehrkräften mit adäquater Ausbildung durchgeführt wird. Darunter verstehen wir ein für das jeweilige Lehramt zugeschnittenes Universitätsstudium der Informatik mit angemessener Tiefe, wie dies für alle anderen Pflichtfächer seit langem selbstverständlich ist. Die Anforderungen an ein solches Studium wurden u.a. von der Gesellschaft für Informatik ausführlich beschrieben (siehe Gesellschaft für Informatik (1999)).

Intellektuelle Ansprüche. Die im folgenden Kapitel 4 vorgeschlagenen Lerninhalte stellen zum Teil hohe Anforderungen an die intellektuelle Leistungsfähigkeit der Schüler. Die Schüler sind zu derartigen Anstrengungen nur in einem Fach bereit und in der Lage, das formal anderen Fächern mit ähnlichen Ansprüchen (etwa Mathematik) gleichgestellt ist.

Kontinuität. Die vorgeschlagenen Lerninhalte sind auf keinen Fall innerhalb einer Jahrgangsstufe vermittelbar. Das Zustandekommen von Wahlkursen hängt aber stark von wechselnden Rahmenbedingungen wie Schülerinteresse, Verfügbarkeit von Räumen und Lehrkräften und ähnlichem ab. Ein kontinuierliches Arbeiten über mehrere Schuljahre hinweg ist deshalb nicht möglich.

Davon unberührt bleiben natürlich *zusätzliche* Angebote aus dem Wahlkursbereich, die insbesondere für Schüler mit speziellen Interessen eine willkommene und fruchtbare Ergänzung des Pflichtunterrichts darstellen können. Ein reguläres Pflichtfach kann jedoch durch solche Kurse niemals ersetzt werden.

3.3.2 Zeitliche Grobstruktur

Die erste Begegnung mit neuen Lerninhalten soll nach den obigen Überlegungen zur Problemorientierung möglichst innerhalb von größeren Unterrichtsprojekten stattfinden. Dazwischen müssen Festigungsphasen eingeschoben werden, in denen der während der letzten Projekte erlernte Unterrichtsstoff systematisiert, eingeord-

net, wiederholt, zusammengefasst sowie direkt und apponiert (siehe Teil A, Kapitel 2) geübt werden kann. In diesen Phasen werden auch die Anforderungen für Leistungserhebungen definiert.

Die tatsächlich verfügbare Stundenzahl je Jahrgangsstufe für ein zweistündiges Fach beträgt ca. 50 Unterrichtsstunden. Je Halbjahr sollten etwa 4 Blöcke mit je 2-stündigen Festigungsphasen eingestreut werden. Damit verbleiben für Projektarbeit ca. 34 Stunden pro Schuljahr, also Raum für 3–4 umfangreiche Unterrichtsprojekte.

3.3.3 Feinstruktur der Projekte

Modellbildung und Simulation als Unterrichtsprinzip in Verbindung mit den bewährten Stufen des unterrichtlichen Vorgehens (siehe Teil A, Abschnitt 4.5.1) sowie die Berücksichtigung der lernpsychologischen Vorgaben aus dem vorangegangenen Abschnitt 3.1 führen uns zu den folgenden Vorschlägen für die Phaseneinteilung der Unterrichtsprojekte (siehe auch Hubwieser u. Broy (1996), (1997)). Die Ähnlichkeit mit der Struktur gängiger Vorgehensmodelle bei der Softwareentwicklung (siehe etwa Rechenberg u. Pomberger (1997)) ergibt sich zwangsläufig aus dem gemeinsamen Prinzip der Problemorientierung. Diese Struktur ist jedoch keineswegs als strenges Schema gedacht. Wir stellen uns eher vor, dass man versucht, bei jedem Projekt alle genannten Phasen zu streifen, wobei deren Reihenfolge u.U. von den Eigenheiten der Aufgabenstellung abhängen kann.

Problembegegnung. Zunächst erfolgt ein erster Kontakt mit einer Problemstellung aus der Praxis, die Notwendigkeit des Einsatzes von neuen Techniken der Informationsverarbeitung wird klar. Ziele dieser Phase sind Motivierung, Veranschaulichung, Wiederholung und Festigung von früher Gelerntem (siehe Teil , Kapitel 2). Der Lehrer wird hierbei naturgemäß im Mittelpunkt stehen, Lehrervortrag, Schülervortrag und Unterrichtsgespräch dominieren. Als Medien sind originale Begegnungen, Film- und Bildmaterial sowie Simulationen am Rechnernetz denkbar.

Informelle Problembeschreibung. Die Schüler versuchen informell, das Problem mit allen seinen Randbedingungen verbal oder graphisch möglichst ausführlich zu beschreiben. Erziehung zu planmäßigem Handeln, Training von Sorgfalt und Umsicht stehen im Vordergrund. Die Schüler arbeiten alleine oder in Gruppen. Als technische Hilfen bieten sich Textverarbeitungs- oder Grafikprogramme an, am besten mit der Möglichkeit der Erstellung gemeinsamer Dokumente über das Schulnetz. In der Softwareentwicklung wäre das Ergebnis der entsprechenden Phase ein Pflichtenheft.

Formale Modellierung. Unter Anwendung der im folgenden Kapitel 4 beschriebenen Modellierungstechniken sollen die Schüler

- Techniken zur Strukturierung von Information erlernen,
- zu Sorgfalt, Genauigkeit, systematischem Denken und Handeln erzogen werden,

- im Hinblick auf ein späteres Studium standardisierte Modellierungsverfahren kennenlernen,
- Einblicke in Methoden zum Entwurf komplexer Informationssysteme gewinnen
- sowie eine Förderung ihrer Abstrahierungsfähigkeit erfahren.

Die Gruppe ist die beherrschende Sozialstruktur, typisch wäre etwa die Entwicklung unterschiedlicher Modellklassen durch einzelne Gruppen mit abschließender gemeinsamer Diskussion der Ergebnisse. Ein willkommenes Hilfsmittel bei der Erarbeitung von Diagrammen wäre ein geeignetes Flow-Chart-Programm, das eine saubere Anordnung der Elemente auf dem Arbeitsblatt, leichte Korrekturen und eine Einbindung in ein Abschlussdokument ermöglicht.

Implementierung und Realisierung. Das Ziel dieser Phase ist es vor allem, die in der vorigen Phase konstruierten Modelle zu überprüfen und zu veranschaulichen. Außerdem trägt die Aussicht auf ein lauffähiges System entscheidend zur Motivation der Schüler bei. Die realisierte Lösung ist dabei immer die Simulation eines mehr oder weniger komplexen Systems. Dies kann sich sowohl auf ein real vorhandenes System als auch auf ein hypothetisches System beziehen. Die Vermittlung profunder Kenntnisse über ein spezielles Programm- oder Programmiersystem wird dabei nicht angestrebt. Es kommen verschiedene Informatiksysteme wie Standardsoftware, Programmiersprachen oder spezielle Simulatoren zum Einsatz, die im folgenden Abschnitt 3.4. besprochen werden. Auch hier sollte die Gruppenarbeit dominieren, etwa in der Programmierung einzelner Module oder in der Realisierung derselben Problemstellung mittels unterschiedlicher Standardsoftware.

Bewertung. Zur Wiederholung, Festigung, Einordnung und Förderung der Kritik- und Urteilsfähigkeit der Schüler folgt abschließend eine Bewertungsphase, ähnlich der Review-Phase der Softwaretechnik. In allgemeiner Diskussion werden die Beschreibungen und Realisierungen bewertet und mit Alternativen verglichen. Es werden Fragen beantwortet wie:

- Was könnte man verbessern?
- Welche Teilprobleme wurden nicht gelöst?
- Wo haben wir das Problem vereinfacht?
- Welche alternative Realisierungen gäbe es?
- Welche ähnlichen Probleme können wir mit unserer Lösung behandeln?
- Welche Fehlerfälle können auftreten, wie reagiert unser System darauf?
- Welche Konsequenzen hat das System auf sein Umfeld im Hinblick auf Datenschutz, Arbeitsplätze, etc.?

3.4 Bemerkungen zu Unterrichtsmedien

Der Begriff „Unterrichtsmedium“ wird zwar häufig gebraucht, ist jedoch sehr schwer einzugrenzen. Meyer (1987) schreibt dazu:

> Was Zweck und was Mittel bzw. Medium des Unterrichts ist, kann also auf unterschiedlichen Ebenen methodischer Reflexion und aus der Perspektive von Lehrern und Schülern je unterschiedlich bestimmt werden. Es gibt keine Medien „an sich", deren theoretischer Status unabhängig von realen Unterrichtsprozessen bestimmt werden könnte, sondern lediglich historisch überlieferte, mehr oder weniger gut begründete Hypothesen über den Mediencharakter bestimmter unterrichtlicher Erscheinungsformen. Deshalb muss der Medienbegriff theoretisch unbestimmt bleiben, auch wenn er in der Unterrichtspraxis durch Sprachkonventionen genau fixiert werden kann.

Diese Unbestimmtheit wird im Informatikunterricht besonders deutlich: Einerseits dient die Behandlung grundlegender informatischer Konzepte auch (und nicht zuletzt) dem besseren Verständnis der Arbeitsweise konkreter Informatiksysteme, die damit in gewisser Weise zum Unterrichtsgegenstand werden, andererseits werden solche konkreten Systeme zum Zwecke der Veranschaulichung von abstrakten Konzepten, also als Medium, eingesetzt.

Zur Veranschaulichung der im Unterricht erstellten Modelle sind alle Medien geeignet, die eine Simulation zulassen. Dabei ist stets im Auge zu behalten, ob der Nutzen des Einsatzes den Aufwand zur Einarbeitung in das System rechtfertigt. Den Schülern muss immer bewusst sein, dass das Ziel nicht im Erlernen der Feinheiten einer bestimmten Programmiersprache oder in der perfekten Beherrschung eines Programmsystems liegt, sondern im Erkennen übergeordneter Strukturen oder Strategien.

Nun sollen einige typische Medien (die aber auch zu Unterrichtsinhalten werden können) vorgestellt werden.

3.4.1 Bürosoftware

Textverarbeitung, Tabellenkalkulationen oder Datenbanksysteme, letztere insbesondere in relationaler Ausprägung, stellen für einfache Beispiele bereits durchaus geeignete Implementierungsmittel dar. Für anspruchsvollere Probleme könnte man makroprogrammierbare Software verwenden. Ideal dafür wären integrierte Programmpakete („Office"-Programme) mit einer durchgängigen, intuitiv verständlichen Makrosprache.

Im Informatik-Anfangsunterricht gewinnt die Untersuchung der typischen Strukturen kommerzieller Softwaresysteme immer mehr an Bedeutung (siehe Knapp u. Fischer (1998), Füller (1999), Hubwieser (1999a)). Mit ihrem sehr interessanten *Dekonstruktionsansatz* versuchen Hampel, Magenheim u. Schulte (1999), diese Unterrichtsbemühungen zu systematisieren und theoretisch zu hinterlegen, indem sie exemplarisch ein spezielles Softwaresystem analysieren lassen:

> Programmierung als Teil des Konstruktionsprozesses eines Informatiksystems steht nicht mehr allein im Mittelpunkt des Informatikunterrichts, sondern ggf. eher am Ende eines Modellierungs-, Formalisierungs-, Abstraktions- und ggf. Dekonstruktionsprozesses. Objektorientierte Modellierung bewegt sich dann zwischen den Ebenen von Systemanalyse (reales Objekt), Formalisierung bzw. Abstraktion von

Aufgaben und Kooperationen (abstraktes Objekt) sowie Klassendefinitionen (implementierte Klasse).

3.4.2 Hypertextsysteme

Anhand von Hypertextsprachen wie der im Internet sehr verbreiteten *Hypertext Markup Language* (HTML) bieten sich gute Gelegenheiten, die Schüler in Techniken der Informationsstrukturierung und -darbietung einzuführen. In der Regel, vor allem in der Unterstufe, wird man unter Umgehung der Sprachsyntax unter Verwendung von entsprechenden Hilfsprogrammen mit den Schülern Hypertextstrukturen aufbauen. Ebensogut können Hypertextsprachen aber auch als Beispiele für formale Sprachen mit einer unmittelbar sichtbaren, „visuellen“ Semantik dienen. Hier kann man die Bedeutung syntaktischer Konstrukte hervorragend veranschaulichen, ohne auf komplexe Zustands- oder Auswertungskonzepte zurückgreifen zu müssen.

3.4.3 Programmiersprachen

Die Implementierung der erstellten Modelle zwingt zumindest gelegentlich zur Verwendung einer Programmiersprache. Welche Aspekte dabei zu beachten sind, soll in Kapitel 4 besprochen werden. In jedem Fall stellt sich die Frage, welche der zahlreichen verfügbaren Sprachen man jeweils verwenden soll, weshalb wir kurz einige Vor- und Nachteile der verschiedenen Sprachparadigmen ansprechen wollen. Die Zuordnung einer Sprache zu einem bestimmten Paradigma ist dabei meist keineswegs eindeutig. So haben z.B. die meisten objektorientierten Sprachen einen stark imperativen Charakter. Dennoch hat jede Sprache ihr „Schwerpunktparadigma“. Wir wollen kurz auf einige wenige Vor- und Nachteile der einzelnen Paradigmen eingehen. Eine ausführlichere Besprechung findet sich z.B. bei Schwill (1995).

Funktionale Sprachen (z.B. Haskell, Gofer, ML). Diese Sprachen haben meist eine sehr knappe Syntax mit der Möglichkeit einer kompakten Darstellung rekursiver Datenstrukturen. Ein Programm ist einfach nur ein Term, in dem selbst definierte Funktionen vorkommen können. Andererseits kann die Wiederholung von Befehlen (unter Einhaltung des Sprachparadigmas) nur über Rekursion gesteuert werden, wodurch jüngere Schüler überfordert sein könnten.

Imperative Sprachen (z.B. Basic, Pascal, C, Modula 2, Oberon). Günstig ist die Analogie zur Funktionsweise des Von-Neumann-Rechners, wodurch ein Grundverständnis für dessen Funktionsweise erzeugt wird. Weniger positiv ist die Verführung zur „Ad-hoc Codierung“ sowie die für Schüler oft schwer durchschaubare Zustandssemantik.

Objektorientierte Sprachen (z.B. C++, Java, Smalltalk). Diese Sprachen liegen nach Schwill (1995) und Füller (1999) nahe am menschlichen Weltbild. Schwie-

rigkeiten bereiten dagegen erfahrungsgemäß häufig die höheren Konzepte der Objektorientierung wie dynamische Bindung, Vererbung oder Polymorphie.

Prädikative Sprachen (z.B. Prolog). Für spezielle Anwendungen wie die Modellierung von Wissensbasen sind diese Sprachen sehr geeignet. Die Syntax ist minimal. Problematisch ist das Grundverständnis der Semantik, das zumindest elementare Einblicke in die Prädikatenlogik voraussetzt.

Wie man sieht, gibt es (derzeit noch) kein optimales Sprachparadigma für schulische Zwecke. Man wird also nicht umhin kommen, im Lauf der Schulausbildung mehrere Sprachen für verschiedene Zwecke einzusetzen. Dabei ist allerdings zu berücksichtigen, dass die Schüler sehr viel Zeit zur Einarbeitung in eine neue Sprache benötigen. Ob sich dieses Problem mit Mischformen wie Logo oder Makrosprachen lösen lässt, muss durch entsprechende Unterrichtsversuche gezeigt werden.

3.4.4 Programmieroberflächen

Neben dem Paradigma der Sprache ist die verwendete Programmieroberfläche von großer Bedeutung für den Unterrichtseinsatz. Hier reicht die Spannweite vom spartanischen Texteditor über spezielle Programmieroberflächen mit Syntaxprüfung bis zur *visuellen Programmierung*, bei der man nur noch die Attributwerte der Objekte setzt. Bei letzterer spart man sich zwar das Eingeben zahlreicher Syntaxelemente, handelt sich jedoch im Gegenzug eine ziemlich undurchsichtige Semantik ein: Warum passiert wo genau was? Welche Information steckt in welchen Attributen und Dateien? Welcher Programmcode wird warum wann ausgeführt (siehe dazu auch Füller (1999))?

Eine schöne Möglichkeit für einen Einstieg in die Programmierung in der Sekundarstufe I bieten beispielbezogene Programmiersysteme wie *Karel* der Roboter (siehe Pattis (1981)). Dabei kann man auf dem Bildschirm einen kleinen Roboter bewegen und zum Auf- und Abbauen von Ziegelsteinbauwerken veranlassen. Dies kann anfangs im Befehlsmodus durch Aufruf entsprechender Kommandos vor sich gehen. Später kann man dann dieselben Befehle zusammen mit grundlegenden Kontrollstrukturen zu einem Programm kombinieren und ablaufen lassen, etwa um ein Haus aufzubauen.

3.4.5 Code-Generatoren und Simulatoren

Eine weitere interessante Perspektive für den Unterricht liefern Code-Generatorsysteme, die aus formalen Beschreibungen auf sehr abstrakter Ebene lauffähigen Programmcode generieren (siehe Broy et al.(1996)). Damit wird den Schülern unmittelbar die Bedeutung ihrer Modellierungsergebnisse vor Augen geführt. An den Hochschulen werden derzeit zahlreiche derartige Systeme entwickelt, die den Schulen in absehbarer Zeit kostengünstig überlassen werden könnten. Durch visuelle Simulation des zeitlichen Ablaufs der entwickelten Modelle

könnte die didaktische Wirksamkeit noch gesteigert werden. Die Schüler hätten dann die Möglichkeit, unmittelbar nach der Modellierungsphase deren Ergebnisse mit der Anforderungsbeschreibung zu vergleichen.

Ein sehr attraktives Beispiel ist der programmierbare Marienkäfer *Kara* der ETH Zürich (siehe Nievergelt (1999)), der direkt über die Eingabe eines endlichen Automaten gesteuert wird.