Propädeutisches Informatikwissen

Susanne Loidl[1], Jörg Mühlbacher[1], Helmut Schauer[2]

[1]Institut für Informationsverarbeitung und Mikroprozessortechnik (FIM), Johannes Kepler Universität Linz, Altenbergerstr. 69, A-4040 Linz
{loidl, muehlbacher}@fim.uni.linz.ac.at
[2]Institut für Informatik (IFI), Universität Zürich, Winterthurerstr. 190, CH-8057 Zürich
schauer@ifi.unizh.ch

Abstract. Auf der Suche nach den langfristig gültigen Grundlagen der Informatik sind in den letzten Jahrzehnten einige Konzepte in den Vordergrund getreten. Eine Konzentration auf die grundlegenden Konzepte, ist gerade im Bereich der allgemeinbildenden Schulen sinnvoll und nötig. Tatsache ist, dass reines Produktwissen zu kurz greift und teilweise bereits schon während der Schulzeit überholt ist. Ein grundlegenderes Verständnis für die Konzepte und deren Zusammenhänge ist daher nicht nur sinnvoll, sondern unumgänglich. Einige dieser „ewigen" Werte der Informatik werden hier kurz dargestellt und dabei aufgezeigt, wie deren Verständnis durch den Einsatz von Applets erleichtert werden kann. Auch wird gezeigt, wie diese bereits jetzt – in Verbindung mit E-Learning – im Unterricht eingesetzt werden.

1 Einleitung

Auf der Suche nach den langfristig gültigen Grundlagen der Informatik – und solche sollten vermehrt in die Lehrpläne allgemeinbildender Schulen Eingang finden – sind in den letzten Jahrzehnten einige Konzepte in den Vordergrund getreten, die hier kurz vorgestellt und diskutiert werden (Eine Auswahl solcher Konzepte wird in [3] vorgestellt). Beispiele dafür sind etwa die Konzepte der Abstraktion insbesondere im Zusammenhang mit Modellierung und Rekursion, die Konzepte unterschiedlicher Notationsformen mit klarer Trennung zwischen syntaktischen und semantischen Aspekten. Bedeutend scheint auch die Unterscheidung zwischen statischen und dynamischen sowie lokalen und globalen Aspekten zu sein. Auch spezielle Eigenschaften von Relationen wie etwa die der Transitivität, Symmetrie oder Reflexivität sowie z.B. der Unterschied zwischen Gleichheit und Äquivalenz sind in der Informatik von grundlegender Bedeutung. Wie und vor allem welche Konzepte vermittelt werden können, soll an folgenden ausgewählten Beispielen vorgestellt werden ([9], [11]).

Nicht diskutiert wird die Frage, in welchem Umfang prozedurale oder objektorientierte Programmierung in die schulische Ausbildung aufzunehmen ist. Programmieren ist

einerseits eine exzellente Schulung im algorithmischen Denken und erfordert anderseits ein Mindestmass an Übung. Letztere zählt zu Fertigkeiten und deren Stellenwert und Umfang hängen vom jeweiligen Schultyp und dessen Ausbildungszielen ab.

2 Beispiele für grundlegende Konzepte in der Informatik

Betrachten wir eine typische Aufgabe der Informationsverarbeitung: die Bestimmung der Masse x eines unbekannten Objektes mittels einer Balkenwaage. Um diese Aufgabe zu analysieren machen wir uns zuallererst ein **Modell** [2].

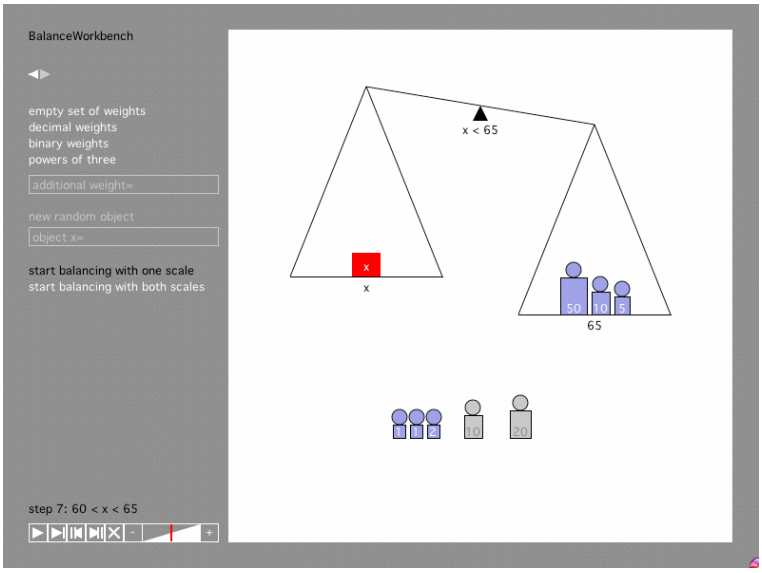

Fig. 1. Bild eines Modells einer Balkenwaage als Beispiel für Modellierung

2.1 Modellierung, Abstraktion, Zustände

Bei der Modellierung kommt es zu einer **Abstraktion**: bestimmte Aspekte der Aufgabe werden als relevant erachtet und in dem Modell berücksichtigt, andere irrelevante Aspekte werden ignoriert. Was in einem Modell als relevant erachtet wird und was nicht ist von fundamentaler Bedeutung und hängt vom Zweck der Modellierung ab. So ignorieren wir etwa die Größe, Form und Farbe des unbekannten Objektes und betrachten die Waage nur im eingeschwungenen Zustand mit drei möglichen Ergebnissen einer Wiegung: die Masse des Objektes auf der linken Waagschale kann kleiner, gleich oder größer der Summe der

Massen aller Gewichte auf der rechten Waagschale sein. Letzteres illustriert die Unterscheidung zwischen **statischen** und **dynamischen** Aspekten der Modellierung sowie das Konzept eines **Zustandes** in dem sich ein System befinden kann. Ein solches System das bestimmte Zustände annehmen kann und diese aufgrund bestimmter **Ereignisse** wechselt, wird als **Automat** bezeichnet. Ist jeder Folgezustand durch den momentanen Zustand und das jeweilige Ereignis eindeutig bestimmt so ist der Automat **deterministisch** und sein Verhalten prognostizierbar. Glückspielautomaten sind typischerweise nichtdeterministisch. Betrachten wir etwa das Plazieren eines Gewichtes auf eine Waagschale als Ereignis, so ist unser Modell einer Waage ein deterministischer Automat. Erlauben wir, dass einmal platzierte Gewichte auch wieder entfernt werden dürfen, so kann ein Zustandswechsel auch wieder rückgängig gemacht werden. Zustandswechsel können somit **reversibel** oder auch **irreversibel** sein. Beispiele für reversible Zustandsänderungen bei Computeranwendungen sind all jene Aktionen die durch **undo** wieder rückgängig gemacht werden können.

Ein weiterer wichtiger Aspekt ist die **Genauigkeit** eines Wiegevorganges. Wir können uns zum Beispiel für ein diskretes Modell mit ganzzahligen Gewichten entscheiden mit dem die Masse des unbekannten Objektes ebenfalls nur ganzzahlig ermittelt werden kann und gleichzeitig offen lassen, ob die Gewichte in Gramm, Kilogramm oder Tonnen angegeben werden. Die Unterscheidung zwischen **diskret** und **kontinuierlich** veränderbaren Werten entspricht dabei einem weiteren für die Informatik grundlegenden Konzept.

Ebenfalls wesentlich für die Modellierung ist die Entscheidung darüber, was an dem Modell fix und was veränderlich ist. So wäre es etwa denkbar, einen Gewichtsatz fest vorzugeben, oder die Gewichte frei wählbar zu machen. Ebenso könnte der Waagbalken konstant immer in der Mitte gelagert sein oder verschiebbar eine frei wählbare Hebelwirkung erlauben. Welches die **Parameter** eines Modells sind und welche Größen als **konstant** und welche als **variabel** betrachtet werden entspricht somit einem weiteren fundamentalen Konzept. Alan Perlis [8]. hat dies bereits vor drei Jahrzehnten mit dem Ausspruch „one man's constant is another man's variable“ vortrefflich kommentiert“.

Ein besonderes Lernziel im Beispiel Waage ist demnach nicht nur das Modellieren in Abhängigkeit von der gewählten Abstraktionsebene, sondern auch eine Diskussion über den Zweck des resultierenden Modells. Für Mathematiker genügt die Gleichung $x*a = g*b$, wenn g das zur Bestimmung von x verwendete Gewicht ist, und a und b die Balkenlängen sind. Dabei ist vorausgesetzt, dass man eine Balkenwaage modellieren will, denn die Gleichung für Waagen, bei der in Abgängigkeit vom aufgelegten Gewicht die Ausdehnung z.B. einer Feder zur Gewichtsbestimmung gemessen wird, ist eine andere.

Informatiker müssen bei einer Verwendung von ganzzahligen Gewichten und deren Größenordnung neben der daraus resultierenden „Ungenauigkeit u“ der Waage den Einschwingvorgang berücksichtigen und werden im Modell daher von einer Ungleichung $|x*a- g*b| < u$ ausgehen bzw. im Modell die Gleichung als erfüllt betrachten, sobald der Winkel α kleiner als ein dem Wiegezweck angepasstes ε ist. Demnach ist die Diskussion des Beispiels Waage geeignet, den Unterschied zwischen formaler, mathematischer

Modellierung und einer Modellbildung aus ingenieurwissenschaftlicher Sicht im Unterricht zu diskutieren.

Ebenso wird die Interpretation von Ergebnissen, die aus einem Modell gewonnen werden, ein Thema.

Besprochen können daher folgende wesentliche Eigenschaften von Modellen:

- Vollständigkeit (in Bezug auf den gewünschten Zweck)
- Widerspruchsfreiheit (Konsistenz
- Verhaltenstreue (in Bezug auf das Original) und die damit verbundene Interpretation der vom Modell gelieferten Daten

Das Metamodell zur Diskussion der Modellierung ist in Fig 2 dargestellt.

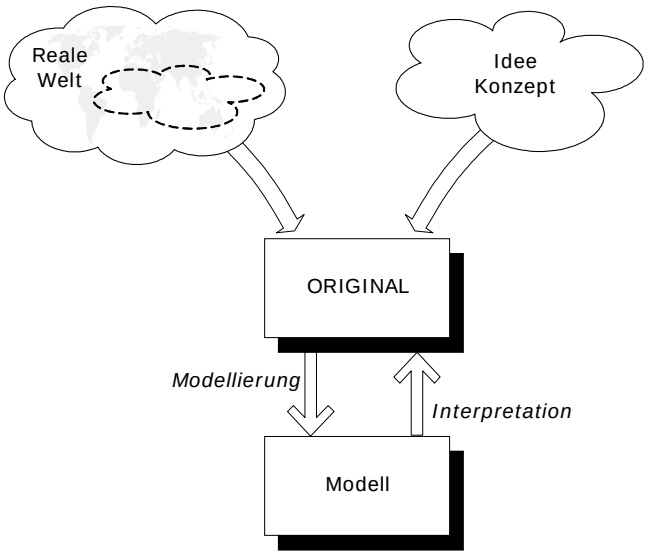

Fig. 2: Metamodell zur Modellierung

Darüber hinaus ist für Informatiker auch der Wiegevorgang ein Thema, was unmittelbar zum Begriff Algorithmus überleitet.

2.2 Algorithmus-Begriff

Der Algorithmus-Begriff gehört unbestritten zu den Fundamenten einer Informatikausbildung. Wie eingangs erwähnt, werden wir zum Programmieren in einer Programmiersprache nicht Stellung nehmen, obgleich Programmieren naturgemäß die besondere Vorgangsweise für Informatiker ist, Algorithmen zu formulieren. An dieser Stelle geht es mehr um den Algorithmus-Begriff losgelöst vom Software-Kontext.

Betrachten wir die Folge von Aktionen die beim Wiegen eines Objektes mittels einer Balkenwaage oder eines zu ihr verhaltentreuen Modells durchgeführt werden. Zum Beispiel können wir rein zufällig solange Gewichte auf die Waagschale legen oder entfernen bis die Waage im Gleichgewicht ist. Abgesehen davon, dass dieses Verfahren zeitaufwendig ist bricht es nur dann ab, wenn die Masse x des Objektes als Summe einer Teilmenge der verfügbaren Gewichte darstellbar ist. (Falls Gewichte auf beide Waagschalen gelegt werden dürfen handelt es sich genau genommen um die Differenz der Summen zweier disjunkter Teilmengen).

Suchen wir daher nach einem zielstrebigen Verfahren, das mit einer möglichst geringen Anzahl von Wiegeschritten die Masse x des Objektes bestimmt. Dies führt uns zum Konzept eines **Algorithmus**. Falls die einzelnen Schritte in einer bestimmten Reihenfolge auszuführen sind, nennt man den Algorithmus **sequentiell**. Algorithmen bei denen die Einzelschritte in beliebiger Reihenfolge oder auch simultan ausgeführt werden können heißen **parallel**. So können zum Beispiel mehrere Gewichte gleichzeitig, und damit parallel platziert oder entfernt werden, die einzelnen Wiegevorgänge hingegen erfolgen sequentiell. Auch die Unterscheidung zwischen sequentiellen und parallelen Abläufen ist in der Informatik von grundsätzlicher Bedeutung, denken wir doch zum Beispiel an die Datenübertragung über serielle oder parallele Schnittstellen.

Die obigen Beispiele sind entsprechend ihrer Formulierung der Klasse der **iterativen** Algorithmen zuzurechnen.

Darüber hinaus stellt sich auch die Frage nach **Rekursion**. Oftmals liegt eine einfache Lösung für ein Problem darin, diese rekursiv anzugehen. Rekursiv bedeutet Selbstbezüglichkeit. Sie tritt immer dann auf, wenn etwas auf sich selbst verweist.

Traditionelle Beispiele wie n- Fakultät: $n! = n*(n-1)!$ mit $n>1$ und $1! =1$ oder die Fibonaccizahlen: $Fib(n) = Fib(n-1)+ Fib(n-2)$ mit $Fib(1) = Fib(2) = 1$ für Rekursion werden sowohl im Informatik-Unterricht als auch im Mathematik-Unterricht angetroffen.

Uns geht es aber mehr um das rekursive Denken, das rekursive Beschreiben von Beobachtungen und das Finden von Problemlösungen durch Rekursion. Je nach Schulstufe müssen verständliche und anschauliche Aufgaben und Beispiele gefunden werden.

Ein erstes einfaches Beispiel einer Rekursion aus dem Alltag ist etwa ein Baum: Stellen wir uns den Querschnitt durch einen Baumstamm vor und betrachten wir die Jahresringe, die sich um sein Zentrum, den Kern, gebildet haben. Ein einjähriger Baum zeigt 1 Jahresring und im inneren davon den Kern. Im allgemeinen Fall sieht man im Querschnitt eines Stamms den äußeren Jahresring, der wiederum den Querschnitt des Baumes enthält als er noch ein Jahr jünger war. Und diese rekursive Sicht setzt sich solange fort, bis das „Abbruchkriterium“ beobachtet wird, der Kern des Baumstammes!

Das folgende Beispiel führt nach unserer Erfahrung im Unterricht durchaus zu einem gewissen Überraschungseffekt, wenn man erklärt, wie man kurz und bündig eine Schlange von wartenden Personen vor einer Supermarktkasse beschreiben kann: eine Schlange Q von Personen P ist entweder leer (eine leere Schlange) oder eine Person P, gefolgt von einer Schlange Q. Wenn man gleichzeitig erläutert, dass man von „Person P“

zu einem beliebigen Objekt abstrahieren kann und ein nichtexistentes „leeres“ Objekt ε in Analogie zur leeren Menge einführt, kommt man zum Begriff der Schlange an sich!

Im Vergleich dazu ist die iterative Beschreibung einer (Warte-)Schlange recht umständlich.

Es hängt vom Ausbildungsziel der Schule und der Schulstufe ab, ob man den nächsten Schritt in Richtung EBNF (Extended Backus Naur Form) an folgendem Beispiel vermitteln will, welches zugleich eine gute Denkübung ist: Es geht um die Beschreibung wie ein Zug aufgebaut ist:

Vereinfacht betrachtet besteht ein Zug durch eine Lokomotive (engine) E am Beginn, die gefolgt wird von mindestens einem Wagon (coach) C.

Fig 3: Ein Zug mit Lokomotive vorne, gefolgt von Waggons

Die rekursive Beschreibung konzentriert sich auf "wie setzt sich ein Zug zusammen": ein nicht leerer Zug (train) besteht aus einer Lokomotive E, gefolgt von einem Endstück (von Waggons) (tail) T bestehend aus Waggons C. Wir können also schreiben:

$$train = ET$$
$$T = C \mid CT$$

Zur Übung kann man dann daraus train = ECC, train = ECCC, train = ECCT usw. ableiten und die Situation bei der Schlange zum Vergleich in Erinnerung rufen: Q= ε|P|PQ.

Ein anderes sehr anschauliches Beispiel, das den Unterschied zwischen iterativem- und rekursiven Problemlösen veranschaulicht ist folgende Aufgabe: Es geht um das Einfärben eines Quadrates unter der Annahme, das zu Färben eines Quadrates der Seitenlänge „1 Pixel“ bereits ein Verfahren bekannt ist.

Die naheliegendste (?) Lösung ist, Zeile für Zeile innerhalb der Quadratgrenzen die Pixels in der gewünschten Farbe zu zeichnen, bis man alle Zeilen des Quadrates auf diese Weise gefärbt hat. Oder man geht stochastisch vor, wählt zufällig Punkte solange innerhalb des Quadrates aus, bis alle solche Punkte (Pixels) eingefärbt sind.

Der rekursive Ansatz liegt auf der Hand:

(1) Hat das Quadrat eine Seitenlänge von“1 Pixel“, so färbt man es mit dem vorgegebenen Verfahren und ist fertig.

(2) Man unterteilt man das Quadrat in vier gleiche, prüft ob (1) bereits angewendet werden kann; andernfalls wendet man auf jedes dieser kleineren Quadrate den Schritt (2) an.

Noch einfacher wird die Aufgabenstellung, wenn man eine Strecke von a nach b auf dem Bildschirm zeichnen möchte: Man halbiert die Strecke und versucht die beiden halbierten Strecken in Analogie zum obigen Beispiel zu zeichnen. Das Abbruchkriterium für die entstehende Rekursion ist erreicht, wenn durch die Teilung nur Strecken der Länge „1 Pixel“ vorgefunden werden, die dann zu zeichnen sind.

2.3 Zeitkomplexität

Zurück zur Waage: Versuchen wir nun den Aufwand für unseren Wiegealgorithmus abzuschätzen: Beim zufälligen Probieren ist die mittlere Anzahl der Wiegevorgänge proportional zur Anzahl aller möglichen Teilmengen der Gewichte. Bezeichnen wir die Anzahl der Gewichte mit n so ist die Anzahl aller Teilmengen der Gewichte gleich 2^n, der mittlere Aufwand steigt daher exponentiell mit n. Der Grund für diesen unnötig hohen Aufwand liegt darin, dass beim zufälligen Probieren die Auswahl der Gewichte bei jedem neuerlichen Versuch unabhängig von den bisherigen erfolglosen Versuchen erfolgt. Die Information, ob bei diesen Versuchen das Objekt leichter oder schwerer als die Summe der platzierten Gewichte war bleibt ungenutzt! Tatsächlich bilden der größte getestete Gewichtswert der leichter als x war die untere Grenze, und der kleinste getestete Gewichtswert der größer als x war die obere Grenze eines Intervalls innerhalb dessen der gesuchte Wert x liegen muss. Die Strategie eines optimierten Wiegealgorithmus kann nun darin bestehen, dieses Intervall bei jedem Wiegevorgang zu halbieren, indem x mit dem arithmetischen Mittelwert der Intervallgrenzen verglichen wird. Ist x leichter als dieser Mittelwert so wird dieser zur neuen oberen Intervallgrenze, ist x hingegen schwerer als dieser Mittelwert so ersetzt dieser die untere Intervallgrenze. Sobald x gleich dem Mittelwert ist oder das Intervall auf die Länge 1 reduziert ist, ist x gefunden. Da jedes Gewicht nur ein einziges Mal platziert wird steigt der Aufwand **linear** mit n. Dieser optimierte Algorithmus ist somit wesentlich effizienter als das zufällige Probieren. Bei diesem (zeitlichen) Aufwand eines Algorithmus spricht man von **Zeitkomplexität**, ein fundamentales Konzept der Informatik. In Zusammenhang mit dem Algorithmusbegriff sind auch noch andere zentrale Fragestellungen, wie die Frage, ob ein Algorithmus hält, ob ein Problem entscheidbar, berechenbar ist, etc. Doch zu diesen Fragen später.

2.4 Zahlensysteme, Code

Da der Aufwand beim Wiegen von der Anzahl n der Gewichte abhängt stellt sich die Frage, ob es gelingt, die Anzahl der Gewichte zu reduzieren ohne den Bereich der damit darstellbaren Gewichtssummen einzuschränken. Mit einem konventionellen Gewichtssatz mit den 8 Werten 1,1,2,5,10,10,20,50 zum Beispiel lassen sich alle ganzzahligen Gewichtssummen im Intervall 0 bis 99 bilden. Diese Auswahl von Gewichten ist offensichtlich vom dezimalen Zahlensystem inspiriert. Interessanterweise hat die Folge 1,2,5,10,20, etc. die originelle Eigenschaft, dass für jedes aufeinanderfolgende Zahlenpaar der linke Wert der ganzzahligen Hälfte des rechten Wertes entspricht (Die Folge 1,2,5,10,20,... entspricht den ganzzahlig gerundeten Werten der E-Normreihe E3 die jeder Dekade drei logarithmisch annähernd äquidistante Werte zuordnet.). Die Folge der Zweierpotenzen 1,2,4,8,16,32,64, etc. folgt ebenfalls diesem Prinzip, mit dem entsprechenden binären Gewichtssatz mit den 7 Werten 1,2,4,8,16,32,64 lassen sich alle ganzzahligen Gewichtssummen im Intervall 0 bis 127 bilden. Obwohl solch ein binärer

Gewichtssatz nicht handelsüblich ist, ist er dem dezimalen Gewichtssatz in jeder Hinsicht überlegen.

Der nachfolgende Screenshot zeigt das Ergebnis eines Wiegevorganges unter Verwendung eines binären Gewichtssatzes. Die auf der Waagschale platzierten Gewichte entsprechen exakt dem Stellenwert der Einsen in der binären Codierung von x.

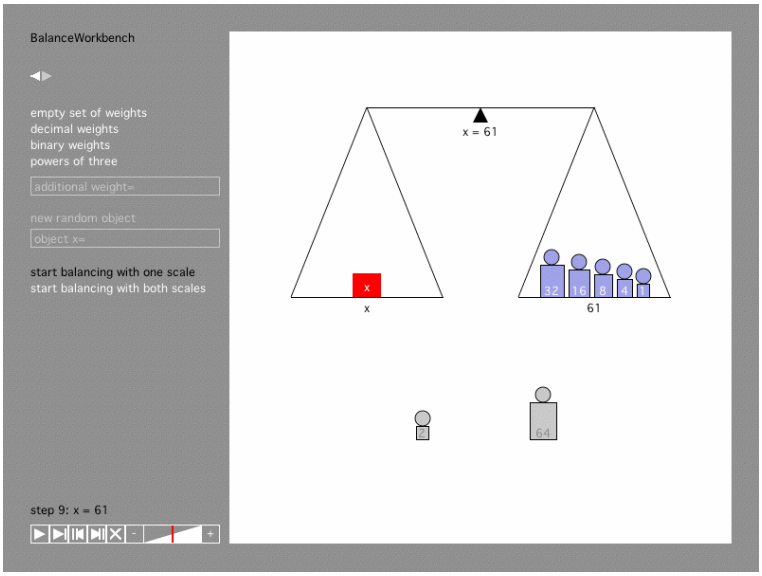

Fig. 3. Binärcode

2.5 Entscheidbarkeit, Berechenbarkeit, NP-Vollständige Probleme

Zu Fragestellungen, ob ein Algorithmus hält, d.h. ob ein entscheidbares, berechenbares Problem, ob ein machbares Problem etc. vorliegt, wendet man gerne ein, dass diese viel zu komplex sind, außerhalb der Unterrichtsziele in Schulen liegt und der Ausbildung auf Universitäten bzw. Fachhochschulen vorbehalten bleiben sollten. Wir möchten in diesem Abschnitt zeigen, dass es sehr wohl einfache und intuitiv formulierbare Beispiele gibt, die diese Themen in einen Informatikunterricht der Sekundarstufe (Informatics in Secondary Schools) einbringen können.

Zugleich damit ist verbunden die strikte Forderung, den Informatikunterricht nur jenen Personen zu überlassen, die über eine einschlägige fachliche Ausbildung verfügen! In diesem Sinne gehen wir davon aus, dass die besprochenen Themen an sich bekannt sind und konzentrieren uns auf den roten Faden für eine adäquate didaktische Aufbereitung.

Zunächst muss die Frage behandelt werden, ob denn alles, was man sich so vorstellt, einer algorithmischen Behandlung und damit letztendlich einer Programmierung unterzogen werden kann.

Als leicht fassbare Problemstellung bietet sich das Halteproblem an: kann man einen Algorithmus angeben, der für jeden beliebigen(!) Algorithmus entscheidet, ob er nach endlich vielen Schritten abbricht oder nicht? Diese Fragestellung lässt sich je nach Vorkenntnissen der Schüler und Schülerinnen relativ leicht verbal beschreiben: man stelle sich jemanden vor, der schon eine Weile auf Ergebnisse wartend vor einem Computer sitzt und sich verzweifelt fragt, ob das ablaufende Programm eben seine Zeit braucht oder ob nicht doch sich ein Fehler eingeschlichen hat, der es als ratsam macht, alles abzubrechen. Der Wunsch nach einem Testprogramm wird laut, doch vorab zu prüfen, ob das betreffende Programm überhaupt jemals anhält und Ergebnisse liefern wird. Dann kann man darauf hinweisen, dass die „Theoretische Informatik“ das vermutlich für Schüler und Schülerinnen überraschendes Ergebnis liefert, dass es in der Tat Aufgabenstellungen gibt, die nicht berechenbar, also nicht programmierbar sind und das genannte Halteproblem solch ein Beispiel ist. Zugleich liefert man den Schülern und Schülerinnen einen guten Grund dafür, dass die Informatik im Fach „Theoretische Informatik“ ihre Grundlagen untersucht.

Im nächsten Schritt kann man davon ausgehen, dass ab sofort nur mehr berechenbare Probleme einer näheren Untersuchung zugeführt werden. Dazu gibt es einfachste und sofort erfassbare Aufgaben wie das Sortieren einer endlichen Menge von Zahlen etc. Zugleich aber wird der Wunsch laut, solche Aufgaben möglichst effizient zu lösen, also sich auf die Suche nach guten Algorithmen zu machen und es genügt vorerst die Eigenschaft „gut“ mit möglichst kurzer Laufzeit zu umschreiben. Zur Illustration eines guten und eines weniger guten Algorithmus nehme man n = 7 ganze Zahlen, ordne sie graphisch einmal als lineare Liste und einmal als binären Suchbaum an und stelle die Frage, wie viele Vergleiche vorzunehmen sind, um herauszufinden, dass eine ganze Zahl x *nicht* in den 7 gewählten n Zahlen enthalten ist: eine erste Begründung für das Fach „Algorithmen und Datenstrukturen“ ist damit geliefert. Will man an dieser Stelle eine Brücke zur Mathematik schlagen und die Vorkenntnisse gegeben sind, bietet sich hier beim aus dem Beispiel resultierenden Binären Suchen der Logarithmus Dualis $\log_2 n$ an. Dabei diskutiert man, wie die Anzahl der Vergleichsschritte anwächst, wenn man statt n nunmehr 2n Zahlen nimmt.

Im nächsten Schritt soll ein besonders eindrucksvolles Beispiel vorgeführt werden, das vor allem auch bewusst machen soll, dass zeitaufwendige Probleme nicht dadurch gelöst werden können, indem man auf den Fortschritt der Technik setzt und sich einen schnelleren Computer besorgt. Zur Illustration dieses Phänomens kann das nachfolgende im Detail besprochene Puzzle Problem herangezogen werden, welches leicht zu erklären ist:

Wir betrachten ein sehr kleines Puzzlespiel der Größe 5x5. Alle Teile sind dabei unterschiedlich, sollen aber, richtig zusammengesetzt, das gewünschte Bild ergeben.

Zuerst muss man im Sinne unseres didaktischen roten Fadens hinterfragen, ob das Problem überhaupt lösbar (berechenbar) ist, d.h. ob bei gegebenen 25 Teilen das Puzzle

gelöst werden kann. Mit dem Hinweis, dass diese Aufgabe für Kinder im Vorschulalter lösbar ist, scheint die Sache zunächst unproblematisch zu sein. Es leuchtet aber ein, dass für einen Computer ein Algoríthmus zu formulieren ist: bevor man sich an das Puzzle-Problem heranwagt, ist zu klären, ob es berechenbar ist! Ein einfacher brute force Algorithmus liefert die positive Antwort dazu:

- Man nummeriere die Stücke von 1 bis 25.
- Man bilde eine Folge aller Stücke. So erhalten wir alle n! Folgen der n (= 25) Zahlen.
- Schließlich ist für jede entstehende Folge zu prüfen, ob sie korrekt ist, also das Puzzle löst.

Wenn man Pech hat („worst case“), findet man die richtige Reihenfolge erst beim n!-ten Versuch!

An dieser Stelle ergibt sich wieder im Sinne eines fächerübergreifenden Unterrichts die Gelegenheit, den Begriff „Permutation“ einzuführen und die Anzahl n! von Permutationen von n Zahlen an wenigen Beispielen abzuleiten oder gar mit Rückblick auf das Thema Rekursion die Definition n! = n(n-1)! zu wiederholen.

Zusätzlich können wir die Stücke auch drehen, wie bei mit einem Puzzlespiel üblich, man erhält die folgende Anzahl von Möglichkeiten:

- Anlegen: 25! = 1,55*10^25
- Drehen: 425 = 1,13*10^15
- Gesamt: = 1,75*10^40

Nehmen wir an, dass man 1 Million Prüfungen pro Sekunde durchführen könnte, dann würde es 1,75*10^34 Sekunden ~ 5,5*10^25 Jahre dauern.

Wenn wir die Drehungen – die Bestimmung der Anzahl der Möglichkeiten könnte auf didaktische Probleme stoßen -weglassen und einen 1000mal schnelleren Computer benutzen, so erhalten wir folgende Zahlen:

Anlegen: 25! = 1,55*10^25 Sekunden, d.h. ~ 4,9*10^11 Jahre warten. Das entspricht immer noch der 15fachen Zeit, die seit dem Urknall vergangen ist! Es hat sich didaktisch bewährt, die Studierenden vorher die Zahl gefühlsmäßig schätzen zu lassen.

Zwei Lehrziele sind damit erreicht: mit der Beschaffung eines schnelleren Rechners ist hier gar nichts gewonnen und man muss sich auf die Suche nach besseren (guten) Algorithmen machen.

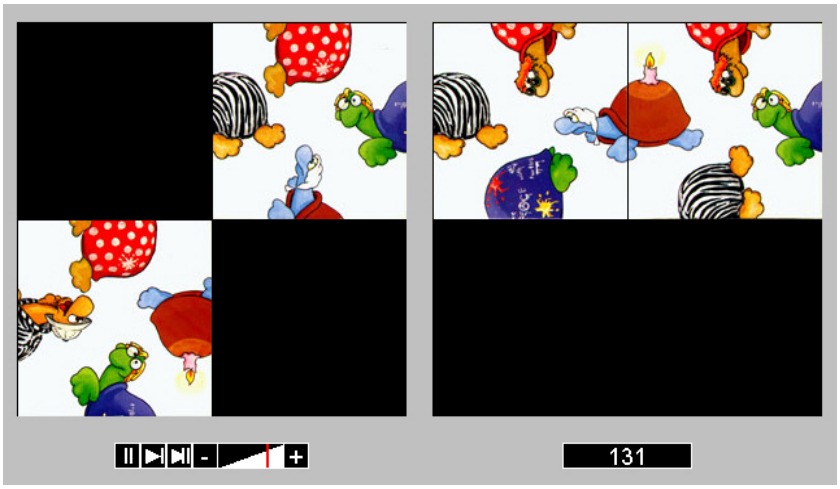

Fig. 4. Puzzle-Problem vereinfacht mit 2x2 Teilen

An dieser Stelle hängt es von der Überzeugungskraft der unterrichtenden Person ab, glaubhaft zu machen, dass für das Puzzle- Problem durchaus Verfahren bekannt sind, innerhalb realistischer Rechenzeit z.B. unter Verwendung von Strukturdaten der Ränder der einzelnen Puzzleteile zu einer Lösung zu kommen.
Eine Diskussion darüber führt aber unmittelbar zur Frage der NP vollständigen Probleme, allerdings im Bewusstsein, dass dieses Thema nur in verbaler und vereinfachter Form erwähnt werden kann. Es ist aber durchaus auch auf dieser Ebene geeignet, bei interessierten Schülern und Schülerinnen entsprechende Neugierde und damit Interesse an der Wissenschaft Informatik zu wecken.

Folgende Beispielswahl hat sich bewährt: man startet mit dem Rundreiseproblem (Besuch von n Städten, ohne eine davon mehr als einmal anfahren zu müssen), welches graphisch mit wenigem Aufwand erläutert werden kann. Ebenso leicht erklären kann man, dass es „berechenbar“ ist, indem wieder alle Permutationen der n Städte angeschrieben wird und für jede einzeln Anordnung geprüft wird, ob sie eine Rundreise ergibt. Im Unterricht der Sekundarstufe bleibt aber an dieser Stelle nichts anderes über, als darauf hinzuweisen, dass interessanterweise (1) für große n noch keine Methode gefunden worden ist, die Aufgabe innerhalb realistischer Zeit zu lösen und (2) bemerkenswerter Weise die Theoretische Informatik folgende Aussagen liefert: (a) es gibt vermutlich gar keinen Algorithmus dafür und (b) man wird nach dem Stande der Wissenschaft aber nie beweisen können, dass die in (a) ausgesprochene Vermutung richtig ist.

Im nächsten Schritt erinnert man die Schüler daran, dass sie zu Beginn eines Schuljahres, wenn ihre Schule viele Klassen hat und viele Lehrer und Lehrerinnen dort unterrichten, kaum einen endgültigen Stundenplan erhalten und dieser immer einen Kompromiss (-> Näherungslösung) darstellt, denn es gilt folgende Aufgabe zu lösen: ein Lehrer kann wohl nicht gleichzeitig in zwei Klasen unterrichten, sollte ohne „Lücken“

aber durchgehend Unterricht haben und die Reihenfolge der Unterrichtsfächer pro Schultag sollte für jede Klasse zweckmässig sein.

Das Bemerkenswerte and diesen beiden (exemplarischen) Aufgabenstellungen ist aber, dass für dieses sogenannte Stundenplanproblem dieselben Vermutungen wie beim Rundreiseproblem bestehen: wenn n (die Anzahl der Unterrichtenden), und m (die Anzahl der Klassen) sehr gross sind, dann kommt man eben mit „Durchprobieren“ nicht mehr weiter. Aber, und das ist das Besondere: würde man dennoch z.B. für das Rundreiseproblem einmal eine „gute“ Lösung – Fachbegriff: polynominalzeitbeschränkt-, finden, so wüsste man, dass auch für das Stundenplanproblem eine in diesem Sinne gute Lösung existiert und es sich lohnte, weiter danach zu forschen. Umgekehrt aber gilt: könnte man trotz gegenteiliger Vermutung beweisen, dass die Aussage (a) beim Rundreiseproblem richtig ist, dann wüsste man, dass auch für das Stundenplanproblem keine entsprechende Lösung existiert. Ferner gilt die Argumentation auch in die andere Richtung: gibt es für das Stundenplanproblem nachweislich keine gute Lösung (Fachbegriff: tractable), so gibt es auch für das Rundreiseproblem keine solche.

Folgende modifizierte Beispielwahl hat sich im Unterricht besonders bewährt: man beschreibt das Stundenplanproblem nur verbal und wählt anschließend das sogenannte Cliquenproblem als weiteres Beispiel für ein NP vollständiges Problem. Dieses kann man wieder sehr einfach durch die Zeichnung eines Graphen [10] mit 5 Knoten und 8 Kanten illustrieren (Fig 5), ohne dass dazu Vorkenntnisse aus der Mathematik benötigt werden. Die Fragestellung lautet: gegeben ein Graph mit n Knoten, gesucht ist der maximale vollständige Teilgraph mit s < n-1 Knoten.

Bei dieser Aufgabe hat man ebenso wie beim Rundreiseproblem die Gelegenheit, auf den Modellbegriff zurück zu kommen: die Knoten entsprechen hier Personen und eine Kante wird eingezeichnet, wenn zwischen zwei Personen eine besondere Beziehung besteht. Eine Clique nennt man eine Teilmenge von Knoten zusammen mit den Kanten, wenn zwischen je zwei Knoten eine Kante existiert. Es ist offensichtlich wie der Konnex zum umgangssprachlichen Begriff „Clique“ ist. Und wieder kann man dieselben Aussagen (1), (2)a und (2)b wie oben treffen.

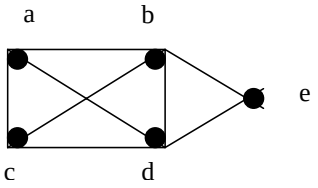

Fig. 5. Graph mit 5 Knoten und einer durch a,b,c,d bestimmten 4-Clique

Unsere Erfahrung bei der Diskussion dieser Fragestellung zusammen mit der Erwähnung, dass es mehr als 1000 zu den beiden genannten Probleme äquivalente [1] gibt, zeigt durchaus eine Bereicherung des Unterrichtes. Der Informatiklehrer und die Informatiklehrerin stehen hier vor ähnlichen didaktischen Problemen wie Kollegen aus Naturwissenschaften, die auf viele offene und nicht gelöste Fragestellungen im Unterricht

hinweisen müssen. Wir halten es durchaus für didaktisch wertvoll, die Grenzen einer Wissenschaft aufzuzeigen, ohne dass im Detail die dahinterstehenden (formalen) Fundamente erklärt werden.

Als besondere Hilfestellung zu diesem Thema wurde der E-Learning Version des Propädeutikum aus Informatik [9] ein „study guide“ hinzugefügt. Man siehe dazu Abschnitt 3.2.

Was einerseits die exakte Definition von „tractable“ (machbar) durch die „groß O“ Notation mit einem Polynom zur Beschreibung der Laufzeitkomplexität betrifft und anderseits die Definition von „zwar berechenbar, aber intractable (nicht machbar)“ betrifft, raten wir davon in der Sekundarstufe ab. Auch wenn der Umgang mit Polynomen vielleicht erwartet werden kann, stößt die Definition von O(f(n)) für die Zeitkomplexität auf Verständnisschwierigkeiten und man sollte davon Abstand nehmen, sofern es sich nicht um besondere Leistungsgruppen der Schüler und Schülerinnen handelt. Daher gehen wir an dieser Stelle auch nicht weiter darauf ein. Diese Themen werden später im Rahmen eines Informatikstudiums genügend tief vorgestellt und sollten diesem vorbehalten bleiben.

2. 6 Information, Sprache, Alphabet

Die **Repräsentation von Information** durch einen Code sowie die Unterscheidung zwischen der Form dieser Repräsentation und ihrer Bedeutung, also zwischen **Syntax** und **Semantik**, sind weitere grundlegende Konzepte der Informatik, die auch die Definition des Informationsbegriffes im Unterschied zu **Daten** und **Wissen** mit einschließen. Eng mit Syntax und Semantik verbunden ist das Konzept der **Sprache** und damit auch der Programmiersprachen, da die Syntax eine Konstruktionsvorschrift dafür ist, wie Wörter einer Sprache zu konstruieren sind, zusammen mit der Festlegung, dass nur derart konstruierte Wörter zur Sprache gehören. Die Semantik legt die Bedeutung dieser **Wörter** einer Sprache fest.

Sprache wiederum bedingt den Begriff des **Alphabets**, da eine Sprache aus einer Menge von Wörtern über dem Alphabet besteht und schließlich ein Alphabet als eine Menge von Symbolen aus einem Zeichenvorrat definiert ist.

Bei der Vorstellung des Begriffes Syntax einer Sprache wird man auch nicht umhin können, zu hinterfragen, wie eine solche denn beschrieben werden soll. Dies führt zur „Metasprache“ und wir erinnern, dass bei der Diskussion von Modellen von einem Metamodell gesprochen worden ist, welches dort durch eine Graphik (Fig 2) realisiert ist! Ferner wurde andeutungsweise bei der Behandlung der Rekursion auf die EBNF hingewiesen, ein Konzept einer Metasprache zur Beschreibung der Syntax.

Klar scheint allerdings zu sein, dass zwar die Themen Alphabet, Code und formale Sprachen in der Sekundarstufe wegen des unmittelbaren Praxisbezuges mit Blick auf Programmiersprachen didaktisch einfach umzusetzen sind, man aber bei Metasprachen wie EBNF doch an die Grenze des Machbaren angekommen sein wird. Will man an dieser

Stelle Programmiersprachen als Beispiele für formale Sprachen überhaupt vermeiden, so bieten sich an ihrer Stelle an: die Regeln zur Schreibweise von syntaktisch korrekten mathematischen Formeln, die in der Chemie verwendete Notation oder die Notenschrift in der Musik. Letztere ist insofern recht zweckmäßig, weil sie auch semantische Annotationen (Lautstärke: piano, forte; Tempo: presto, usw.) umfasst!

2. 7 Relationen

Selbstverständlich gehört auch eine **Klassifizierung** von Daten bezüglich ihrer Eigenschaften, ihrer **Struktur** und ihren **Relationen** zu den langlebigen Konzepten, mit denen Eigenschaften wie
Symmetrie oder **Äquivalenz** und damit **Äquivalenzklassen** erläutert werden können.

Die Aufzählung dieser Konzepte ist rein exemplarisch, beileibe nicht vollständig und gibt (hoffentlich) Anlass für weitere Diskussionen. Sie soll jedoch zeigen, dass ein allgemeinbildender Informatikunterricht nicht zufällige technologische Artefakte sondern langlebige Konzepte zum Inhalt haben muss und zugleich fächerübergreifend wie hier zur Mathematik gestaltet werden kann (oder umgekehrt).

3 Möglichkeiten der neuen Medien und ihr Einsatz

Die einzelnen Abbildungen zeigen nun bereits auf, in wie fern die neuen Medien bzw. E-Learning helfen können, diese „ewigen Werte“ der Informatik besser darzustellen. Am FIM und auch am IFI ist E-Learning bereits seit Jahren ein wichtiges Thema. Dabei liegen die ersten Schritte in diese Richtung am FIM bereits 20 Jahre zurück. Damals wurden bereits CBT (Computer Based Training)- Kurse (zum Thema Programmieren, Betriebssysteme u.a.) entwickelt und unterstützend zur traditionellen Lehre angeboten.
In technischer Hinsicht gibt es jetzt dem Stand der Technik entsprechend mehr Möglichkeiten als damals. Auch ist die Skepsis, die früher sehr heftig den Konzepten eines computergestützten Unterrichtes entgegengebracht worden ist, mittlerweile zumindest einer differenzierten Betrachtungsweise gewichen.

3.1 Zur Entwicklung

Aus dem Schwerpunkt E-Learning heraus sind mehrere Werkzeuge entstanden, die auch seit einigen Jahren im Unterricht eingesetzt werden. Konkret hat dabei das FIM das WeLearn.Framework entwickelt, dessen Umfang kontinuierlich erweitert wird und dass derzeit folgende Komponenten umfasst:

- WeLearn.System: Eine offene, universell einsetzbare und einfach bedienbare E-Learning Plattform . WeLearn ist web-basiert und erfordert auf der Seite der Benutzer einzig einen Webbrowser.
- WeLearn.DidacticalTemplates: Didaktische Modelle für den universitären, schulischen und Erwachsenen-Fort- und Weiterbildungsbereich, ferner zur Unterstützung der Unterrichtsgestaltung in Informatik in den oberen Klassen der Sekundarschulen.
- WeLearn.Tools: Werkzeuge, wie einen Offline-Konverter, der aus bestehenden Online-Kursen automatisch eine Offline-Präsentation derselben erstellt und dadurch eine Verbreitung von Lehr- und Lernmaterial auf CD-Rom, DVD erlaubt, etc.
- WeLearn.Courses: Elektronisches Kursmaterial, speziell zur Umsetzung unserer Vorstellungen über Propädeutisches Informatikwissen, wie es in dieser Arbeit exemplarisch vorgeschlagen worden ist.

Wir verweisen dazu auf [5],[7] und [9]. Ferner wurde in einer Studie die Akzeptanz des Lernmaterials und der verwendeten Welearn.DidacticalTemplates untersucht [6], die ermunternde Ergebnisse brachte und dazu führte, dass diese Templates auch für den Unterricht zum Thema Betriebssysteme, Netzwerke und Verteilte Systeme sowie Sicherheitsfragen in der Informatik Anwendung finden.

3. 2 Das Propädeutikum aus Informatik

Als wesentlichen Bestandteil zur Umsetzung unserer Vorstellungen über propädeutisches Informatikwissen sehen wir speziell aufbereitete Lehr- und Lernmaterialen an, die sowohl über die Plattform WeLearn verfügbar gemacht werden, als auch auf CD- Basis den Studierenden angeboten werden Das „Propädeutikum aus Informatik“ ist eine Einführungslehrveranstaltung für Studierende der Informatik, die vom FIM an der Johannes Kepler Universität Linz gehalten wird. Es wird regulär im Wintersemester abgehalten, wobei hier als didaktisches Modell ein Blended Learning [4] Verwendung findet: Präsenzphasen wechseln sich mit selbstorganisierte L Lernabschnitten ab. Im Sommersemester wird mit Blick auf Berufstätige, sonstige Nachzügler und für (!s.u.) interessierte Schüler in den oberen Klassen von Sekundarschulen der Stoff wieder angeboten, allerdings nach einem gemeinsamen KickOff Meeting ausschließlich auf Basis von Distance Learning.

Darüber hinaus wird das Propädeutikum nicht nur an der JKU Linz angeboten, sondern auch – in eine andere Lehrveranstaltung eingebettet- an der Universität Zürich, wo Studierende der Wirtschaftsinformatik unter Verwendung desselben elektronischen Unterrichtsmaterials mit den hier diskutierten Themen vertraut gemacht werden. Ebenso wurden Teile davon in einem Lehrgang akademischen Charakters an der FH Vorarlberg erfolgreich verwendet.

Das elektronisch verfügbare Material umfasst derzeit:

- Einen Studyguide: Eine Anleitung für selbstbestimmtes Lernen und eine Erläuterung von Teilen des Lernstoffs, aufbereitet als Dialog zwischen Jugendlichen. Zielgruppen sind insbesondere Schüler der oberen Sekundarstufe

- Den Lernstoff insgesamt als illustrierte, teils interaktive HTML-Seiten
- Den Lernstoff als Langtext, wie er auch als klassisches Skriptum in gedruckter Form vorliegt und den Lernenden zur Verfügung steht .
- Die gesamten Vortragsfolien für einzelne Präsenzphasen. Zielgruppe sind die regulär im 1. Semester für Informatik inskribierten Hörer im jeweiligen Wintersemester.
- Self-Assessment: Übungen und eine Musterklausur, damit Lernende ihren Lernfortschritt überprüfen können und dabei Rückmeldung bekommen, welche Teile der Lerninhalte noch eine intensiveren Auseinandersetzung bedürfen.
- Lernapplets, auf Basis derer die Lernenden selbst Experimente und Simulationen durchführen und sich den Lerninhalten selbst „erfahren“ können Die im Kapitel 2 besprochenen Applets sind unter anderen dort zu finden.

Mit Blick auf den Unterricht in vorgeschalteten höheren Schulen ist folgendes wichtig:
Parallel dazu wird das elektronische Material an Höhere Schulen (AHS, Sekundarstufe) ausgegeben, die es dort für den Informatikunterricht bzw. die Vorbereitung auf eine Matura in Informatik verwenden können (s.u.). Im Informatikunterricht ausgewiesene Sekundarstufenlehrer verwenden das auf CD verfügbare E-Learning Material im Unterricht oder haben selbst eine WeLearn Server, der an Österreichischen Schulen lizenzfrei ist, installiert und stellen über diesen nicht nur das Lernmaterial zur Verfügung, sondern betreuen auch ihre Schüler und Schülerinnen zusätzlich bei auftretenden Fragen über eingerichtete Diskussionsforen. Bemerkenswert dazu ist auch folgende geltende Beschlusslage der JK- Universität Linz: Schülern und Schülerinnen, die in Informatik eine Matura abgelegt haben und wenn der Maturastoff in etwa dem in dieser Arbeit vorgestellten propädeutischen Grundlagen entspricht, brauchen bei einem nachfolgenden Informatikstudium an der JKU das Propädeutikum aus Informatik nicht nochmals belegen.

4 Zusammenfassung und Ausblick

Unsere Zeit wird nachgesagt, dass sie besonders schelllebig ist, und gerade für die noch „junge“ Disziplin der Informatik trifft dies im besonderen Maße zu. Wenn wir davon ausgehen, dass die Informatik in den sechziger Jahren des 20. Jahrhundert ihren Durchbruch hatte, so sprechen wir von nicht einmal 50 Jahren Informatik im Gegensatz zu ein paar tausend Jahren auf die beispielsweise die Mathematik in ihrer Geschichte zurückblicken kann. Umso rasanter ist jedoch die Entwicklung der Informatik verlaufen und blickt man dabei vor allem auf den Bereich der Software, so werden deren Produkte immer zahlreicher und die Halbwertszeit derselben immer geringer. Eine Konzentration auf die grundlegenden Konzepte (wie sie hier beispielhaft dargestellt wurden), scheint daher gerade im Bereich der allgemeinbildenden Schulen für sinnvoll und nötig. Tatsache ist, dass reines Produktwissen und bloße Fertigkeiten zuwenig und auch teilweise bereits schon während der Schulzeit überholt sind. Ein grundlegenderes Verständnis für die Konzepte und deren Zusammenhänge ist daher nicht nur sinnvoll, sondern unumgänglich.

Referenzen

1. Garey M.R., Johnson.D.S.: Computers and Intractability: A guide to the Theory of NP-Completeness, W.H. Freeman, San Francisco, 1979
2. Hubwieser P.: Modellierung in der Schulinformatik. LOG IN 19, Heft 1. S.24-29, 1999
3. Informatik als Grundbildung; Informatik Spektrum, Band 27, Heft 2-4, 2004
4. Loidl, S.: The Beautiful but challenging World of Elearning. In Auer, M. E. and Auer, U., editors, International Conference on Interactive Computer Aided Learning, The Future of Learning, Villach Austria, Kassel university press 2004, ISBN 3-89958-089-3
5. Loidl, S., Sonntag, M.: Using metadata in creating offline views of e-learning content; in: Auer, M., Auer, U. (Ed.): ICL; Learning Objects and Reusability of Content, Kassel university press 2003
6. Mühlbacher, J., Mühlbacher, S.C., Loidl, S.: Learning Arrangements and Settings for Distance Teaching/Coaching/Learning: Best Practice Report. In Hofer, C., Chroust, G. (Ed.) IDIMT – 2002
7. Paramythis, A., Loidl, S.: Adaptive Learning Environments and e-Learning Standards; in: Roy Williams (Eds.): Proceedings of the 2nd European Conference on e-Learning, Glasgow, 2003
8. Perlis, A.J.: Epigrams on programming. SIGPLAN Notices, 17 (9),1982
9.Propädeutikum aus Informatik, http://welearn.fim.uni-linz.ac.at, 2004
10. Rosen, K.H.: Discrete Mathmatics and its Applications, 5th Edition, McGraw-Hill, 2003
11. Schauer, H.: Langlebige Standards in einer schnelllebigen Welt, CD Austria, 5/2004