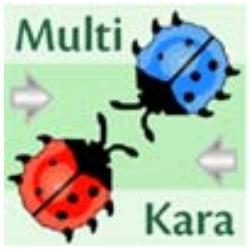
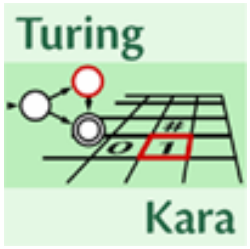
Lernumgebungen fürs Programmieren: Karas... und die Turtles

Raimond Reichert – raimond.reichert@acm.org





beta

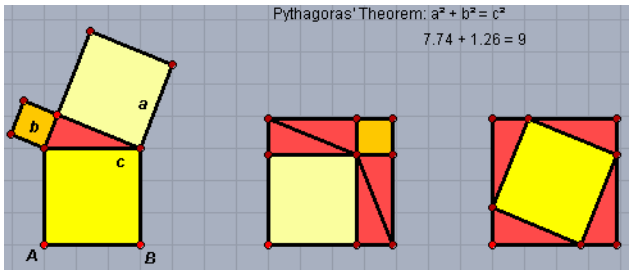
Inhalt

Math education for „math users“

$$a^2 + b^2 = c^2$$

Many professions require the application of mathematical methods and tools.

The professionals do not need to prove results!



Yet as part of our general education, we all learn about the concept of proof!

Programming education for computer users

Programming as part of general education:

we all depend on computers

it is hard to get a feeling for how they work without understanding the concept of „program“

the concept of „specifying processes that evolve over time“ is fundamental

Eine Kernidee der Informatik: Computer ≅ formales System

Intuition

Formale Beschreibung

Das Ziel: Algorithmisches Denken

It has often been said that a person does not really understand something until he teaches it to someone else. Actually a person does not *really* understand something until he can teach it to a *computer*, i.e., express it as an algorithm. [...]

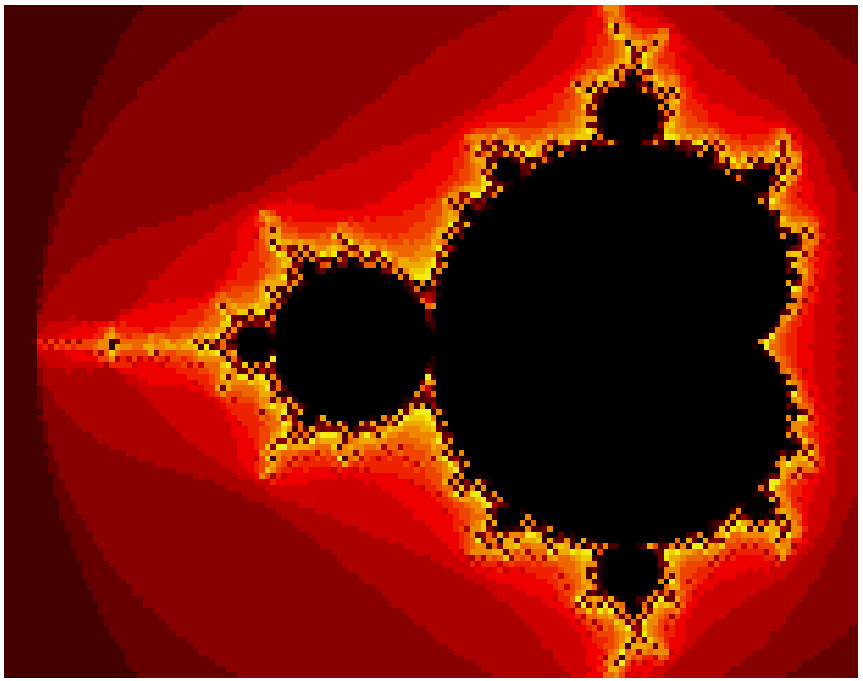
The attempt to formalize things as algorithms leads to a much deeper understanding than if we simply try to comprehend things in the traditional way.

Knuth: *Computer Science and its relation to mathematics*.
The American Mathematical Monthly, 81(4), 1974.

Inhalt

Wie einsteigen ins Programmieren? So sicher nicht!

```
int m,u,e=0;float
l,_,I;main(){for(;1863-
e;putchar((++e>923&&952>e?60-m:u)
["\n)ed.fsg@eum(rezneuM
drahnreB"]))for(u=_=I=0;80-
(m=e%81)&&I*I+_*_<6&&20-
++u;_=2*I*_+e/81*.09-1,I=I)I=I*I-_*_-
2+m/27.;}
```



Wie einsteigen ins Programmieren?

Verwendung professioneller Programmiersprache?

```
import JavaKaraProgram;

public class FollowWall extends JavaKaraProgram {

  public void                {
    while (true) {
      if (kara.treeFr        && kara.treeRight()) {
        kara.turnLe
      }
      else if (!      .treeFront())
        if (kara.treeRight()) {
          kara.m      ;
        }
        else {
          kara.turnRight();
          kara.move();
        }
      }
    }
  }
}
```

?

C++

almost 5000!!

Java

Programmiersprachen: Wie Sand am Meer

Mother Tongues

Tracing the roots of computer languages through the ages

Just like half of the world's spoken tongues, most of the 2,300-plus computer programming languages are either endangered or extinct. As powerhouses C/C++, Visual Basic, Cobol, Java and other modern source codes dominate our systems, hundreds of older languages are running out of life.

An ad hoc collection of engineers-electronic lexicographers, if you will-aim to save, or at least document the lingo of classic software. They're combing the globe's 9 million developers in search of coders still fluent in these nearly forgotten lingua frangas. Among the most endangered are Ada, APL, B (the predecessor of C), Lsp, Oberon, Smalltalk, and Simula.

Code-raker Grady Booch, Rational Software's chief scientist, is working with the Computer History Musuem in Silicon Valley to record and, in some cases, maintain languages by writing new compilers so our ever-changing hardware can grok the code. Why bother? "They tell us about the state of software practice, the minds of their inventors, and the technical, social, and economic forces that shaped history at the time," Booch explains. "They'll provide the raw material for software archaeologists, historians, and developers to learn what worked, what was brilliant, and what was an utter failure." Here's a peek at the strongest branches of programming's family tree. For a nearly exhaustive rundown, check out the Language List at *HTTP://www.informatik.uni-freiburg.de/Java/misc/lang_list.html*. - Michael Mendeno

Key

1954 Year Introduced
Active: thousands of users
Protected: taught at universities; compilers available
Endangered: usage dropping off
Extinct: no known active users or up-to-date compilers
Lineage continues

1954 1955 1956 1957 1958 1959 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001

Fortran — Once one of the most widely used languages in science and engineering. — Fortran IV — Created for the IBM 7090/94 — Fortran 90 ISO/IEC

Flow-Magic — Originally B-O. One of the first English-like data-processing languages. — Cobol — The culprit behind many Y2K bugs. Also known as Common Business-Oriented Language. — Cobol 61 — ANSI Cobol 74 — ANSI Cobol 85 — OO Cobol

Scheme — A Lisp-like language designed by MIT researcher Seymour Papert to make programming accessible to children through graphics and simple syntax. — A Lisp offshoot that's still used in introductory programming classes at universities. — Scheme 84 — Scheme IEEE — Scheme R5RS

Haskell 1.0 — Haskell 1.1

George Radin's attempt to combine the best features of Fortran, Cobol, and Algol 60.

Logo — Object Logo — Clos — Common Lisp Object System.

IPL — Possibly the first list-processing language used in AI guru Ed Feigenbaum's PhD thesis.

Lisp — Invented by John McCarthy at MIT, Lisp has an unusual syntax made up of lots of nested parantheses. Still popular with AI researchers. — Common Lisp — ANSI Common Lisp

PL/1 — Algorithmic Language, designed as a portable language for scientific computations. Algol 68's complexity was either ahead of its time or rightfully doomed, depending on whom you ask. — ANSI PL/1

PL/M — Programming Language/Microcomputers

Self — A simple object-oriented language. — Self 4.0

The version Xerox released to the public.

NetRexx

Initially called Oak. Still one of the fastest-growing languages around, despite the standards fued between Sun and Microsoft. Somewhat like C++, Java allows for "write once, run anywhere" portability across the Net.

Smalltalk — The first language to show off the power of object-oriented code. Developed by Software Concepts Group, Xerox PARC, led by Alan Kay. — Smalltalk-80

Algol 58 — The hoped-for Esperanto of the computing world. Designed by an international Zurich-based committee as a universal language, it was one of the first attempts at making software more portable. Initially called International Algebraic Language. — Algol 68

Rex 1.00 — Rex 2.00 — Restructured Extended Executor — Object Rexx

Simula — Popular in Europe during the '70s, Simula introduced the now-standard concept of object-oriented, rather than procedural, programming. Designed by Kristen Nygaard and Ole-Johan Dahl. — Simula 67 — All-purpose version of Simula

An object-oriented language. — A simple language (A = argument, B = basic value, C = ?). — Interactive object-oriented language. — Java — Java 2(v1.3)

Prolog — Programming on Logique for natural-language processing. Popular for AI programs.

Clu — Called Cluster, it's an object-oriented academic language created to teach rigid engineering skills.

Marco and text-processing language used by Unix admins to manipulate large config files programmatically, rather than editing them by hand.

Objective-C — Eiffel — Eiffel 3 — Microsoft's answer to Java, it is a key component of the Microsoft.Net platform for Web services.

New AWK, a pattern scanning and processing language. — ABC — Sather 0.1 — Ruby — An object oriented language.

AWK — NAWK — Python — Python 1.6

A popular language among Web site builders, it includes features missing from Perl. — ECMA Script

Possibly the most common language today. Adds object-oriented features to C. — Tool Command Language, "tickle." The duct tape of programming (a scripting language for patching together different languages). — Cmm — JavaScript — Microsoft's version of JavaScript. Can't they share anything? :-) — C#

C-, a scripting language. — JScript

One of the most widely deployed languages today. Windows and Unix OSes are written mostly in C and its descendants. — C with Classes — C++ — Internet C++

Concurrent C — Tcl — Tcl/Tk — Tcl/Tk 8.1

BCPL — B — C — ANSI C (C89) — ISO C (C95) — Updates C++ for the Net with a Java-like virtual machine, so code can run on any kind of computer.

Basic Combined Programming Language — Used primarily for non-numeric programming.

ADA — ADA 83 — ADA 95

The US Department of Defense's effort to craft a standard object-oriented language for its work. Named after Ada Lovelace, arguably the world's first computer programmer, and created by Jean Ichbiah's team of Honeywell.

Modula 3 — Found in millions of Web pages. Originally dubbed LiveScript, it was renamed by Netscape marketers who licensed the name to ride Java's buzz. It has little in common with that language.

A teaching language named for French mathematician Blaise Pascal. Designed by legendary Swiss programmer Niklaus Wirth for simplicity, in reaction to the complexity of Algol 68.

Modula — Modular Language — Modula 2 ISO

Object-oriented Pascal, designed for simplicity. — Object Pascal — Oberon — Another creation of Niklaus Wirth. Later modified by Robert Griesemer for numerical apps on supercomputers. — Delphi

Pascal

Csh — The swiss Army Knife of programming (a k a Practical Extraction and Report Language), used for patching together different languages. Spawned a quasi-literary culture that writes Perl haiku — Perl 1.000 — Perl 4.000

String-processing language for text and formula manipulation, common in text processors. Invented by David Farber, Ralph Griswold, and Ivan Polonsky at Bell Labs.

Programmer's easy street - with this descendant of Snobol, there's no need to know the underlying OS.

sh — Ksh

A general-purpose language known for its beauty and grace. Designed by Ralph Griswold to be successor to Snobol4.

C-Shell, a scripting language and command-shell interpreter. Written by C programmers to make Unix command lines more like C's. — Standard ML, a general-purpose language. — The kitchen sink of command-line programming features for Unix, based on the original Bourne shell (a k a Bourne Again Shell). — Bash — Frame-based language.

King of the one-liners, also referred to as A Programming Language. An algebra-like language with a special set of characters for writing complex programs as short formulas. Designed by Kenneth Iverson at Harvard.

Snobol — Snobol4 — Icon — ML — Meta Language — SML — SML 90 — SML 97

Page-description language for printers and graphics sistems; 75 percent of all commercial documents are produced on PostScript printers.

Word-based language first used to guide the National Radio Astronomy Observatory telescope at Kitt Peak in Arizona.

Caml — Caml 2-6.1 — Objective Caml — O Caml 2 — O Caml 3.00

Forth — PostScript — PostScript Level 2 — J

Although mocked by "real" programmers for its limited abilities, Basic has outlived many more advanced languages, as well as the RadioShack TRS-80 computers that made it a household word. Stands for Bigenner's All-Purpose Symbolic Instruction Code.

APL

Categorical Abstract Machine Language

Survival of the Fittest

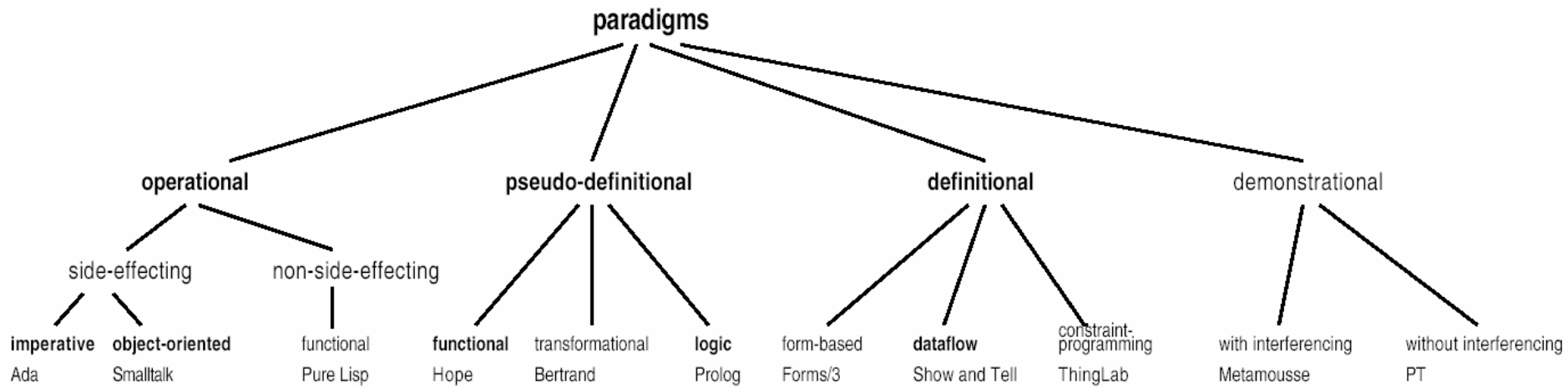
Reasons a language endures, with examples of some classic tongues

- **Appeals to a wide audience** C (bolstered by the popularity of Unix)
- **Gets a job done** Cobol (designed for business-report writing)
- **Delivers new functionality** Java (runs on any hardware platform)
- **Fills a niche** Mathematica (speeds up complex computations)
- **Offers a modicum of elegance** Icon (has friendly, line-oriented syntax)
- **Has a powerful user base or backer** C# (developed by Microsoft for .Net)
- **Has a charismatic leader** Perl (programmer-author Larry Wall)

Sources: Paul Boutin; Brent Hailpern, associate director c

http://www.digibarn.com/collections/posters/tongues/

Programmierparadigmen: Eine überschaubare Anzahl



Ada	Ada-95	Lisp	Hope	Prolog
Basic	C++			
C	Delphi			
Cobol	Java			
Modula	Oberon			
Pascal	Smalltalk			

Klassischer Ansatz in Lehrmitteln

1 The scope of Java
2 A First Java Program
3 Introductory Graphics
4 Variables and Calculations
5 Methods
6 Events
7 Decisions - if and switch
8 Repetition: while, for and do
9 Objects and Classes
10 The User Interface
11 Inheritance
12 Calculations
13 Arrays - one dimensional
14 Arrays - 2D
15 Strings
16 Exception handling
17 Creating a user interface
18 Stand-alone applications
19 Files
20 Graphics and Sound
21 Object-Oriented Design
22 Program style
23 Testing
24 Debugging
.....

Die Gefahr beim klassischen „strukturellen“ Ansatz

Wenn schon ein „Hello World“ so kompliziert ist...

```
import java.applet.Applet;
import java.awt.Graphics;
public class HelloWorld extends Applet {
  public void paint(Graphics g) {
    g.drawString("Hello world!", 50, 25);
  }
}
```

... lernen die Schüler eine konkrete Programmier-sprache von ihrer Syntax her

... können Schüler mit Spezialwissen auftrumpfen

... bleiben die grundlegenden Konzepte schnell einmal auf der Strecke

... bleibt wenig Raum für Gedanken zur Modellierung und zu Problemlösestrategien

Inhalt

Wie überwindet man im Unterricht die Anfangsschwierigkeiten ?

Mit einer ganz kleinen und einfachen Sprachumgebung starten

Programmierumgebung muss Visualisierung unterstützen und auf verschiedenen Stufen der Repräsentation ansetzen

Konzepte müssen an sinnvollen, konkreten, alltagsnahen Beispielen gezeigt werden

Konzepte, die verschiedene Paradigmen und auch kognitive Präferenzen berücksichtigen

Einführung ins Programmieren: Drei Ansätze

Incremental, traditional approach

Folge von wachsenden Teilmengen der Spache.

Sub-language approach

In sich abgeschlossene Teilmenge der ganzen Sprache mit in der Regel einfach zu visualisierenden Operationen (z.B. Turtle Graphik als Teilmenge von Logo).

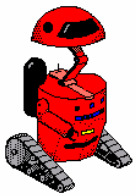
Mini-language approach

Künstliche Miniatursprache mit eigenen Befehlen und eigener Syntax (z.B. Karel the Robot).

Einführung ins Programmieren: Mini-Sprachen

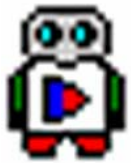
LOGO-turtle

~ 1970



Karel, the robot

~ 1980



Karel++

~ 1990

trend: more complex!

Mini-languages have many advantages:

small language, small syntax, simple semantics

visualisation of some actor in some world

visualisation of program execution

integrated development environment

Einführung ins Programmieren: Mini-Sprachen

Programming model complexity

Object-oriented programming language

Procedural programming language

for example

Karel++

for example

Karel, the robot

more complex

Einführung ins Programmieren: Mini-Sprachen

Most mini-languages are based on real-world programming languages.
⇒ Inherently complex!

The theory of computation focuses on developing simplest possible models for various purposes.

Use a simple model of computation as the means of programming in a mini-environment.

Programming for novices, based on a simple model of computation

Programming model simplicity

Object-oriented programming language

Procedural programming language

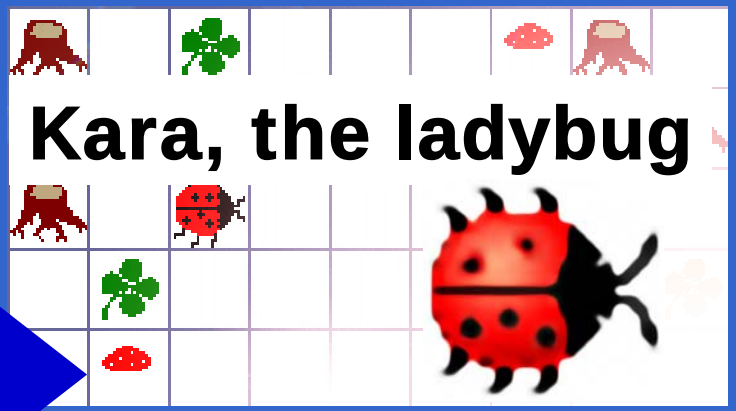
for example

Karel++

for example

Karel, the robot

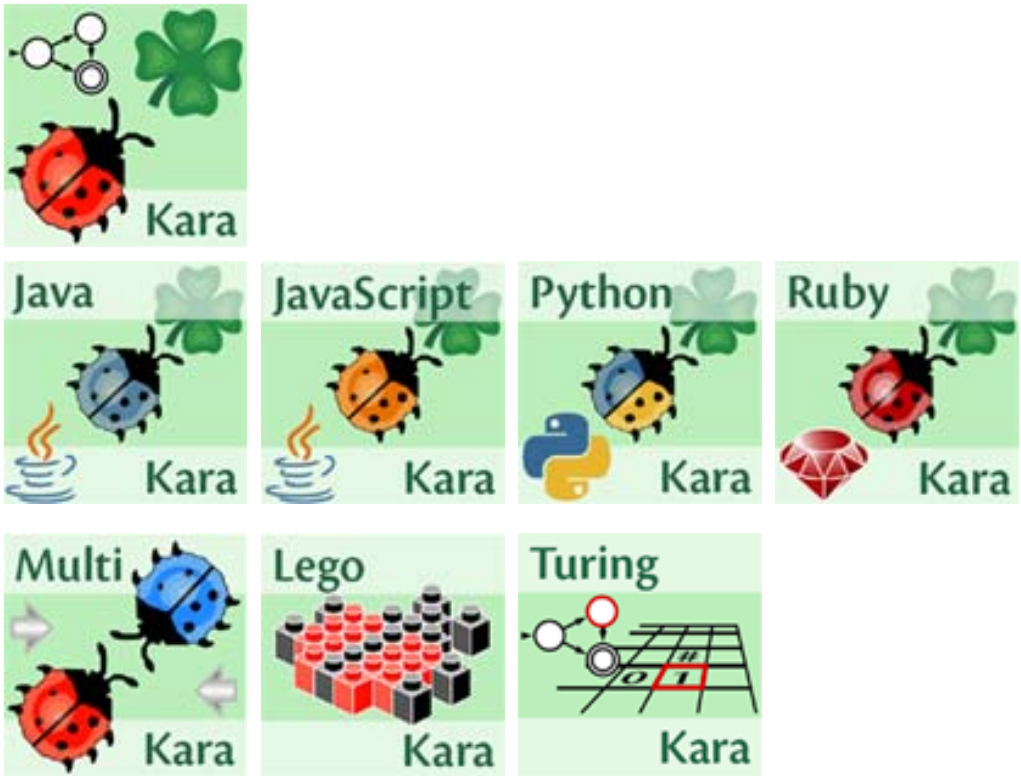
Finite state machine



less complex!

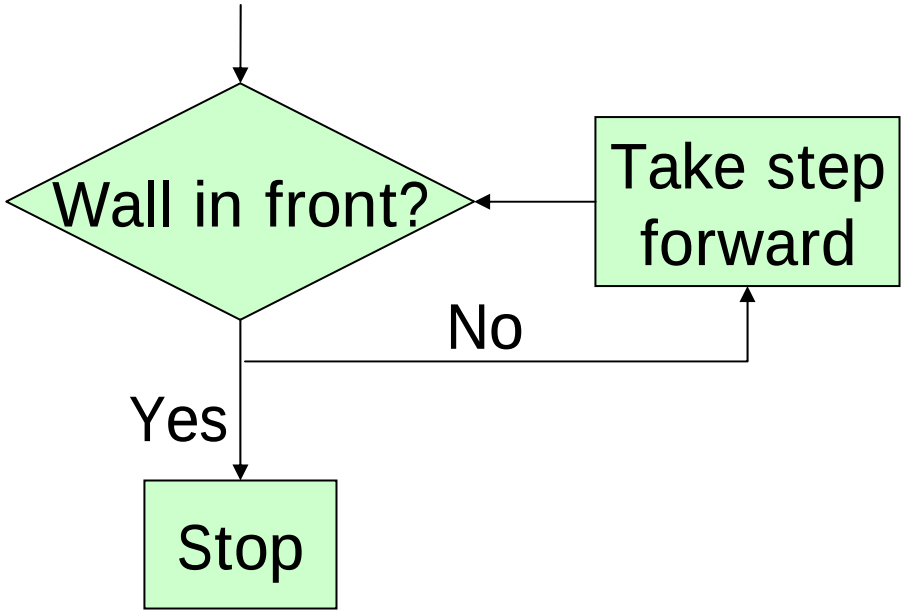
Inhalt

Lernumgebungen fürs Programmieren: Theorie-basierter Einstieg mit Kara



Raimond Reichert - raimond.reichert@acm.org

Das Ziel der Kara-Umgebungen: Reduktion auf das Wesentliche



Die Anforderung: Für didaktische Zwecke soll das Programmiermodell so einfach wie möglich sein.

Idee: Endliche Automaten für den Einstieg ins Programmieren

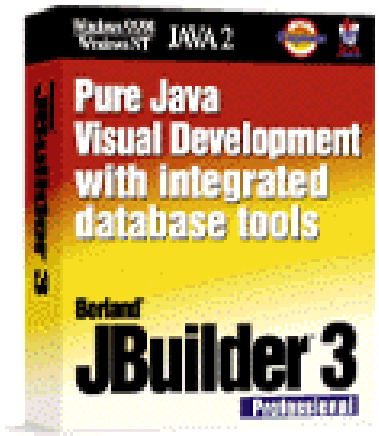
Endliche Automaten...

... sind alltagsnah

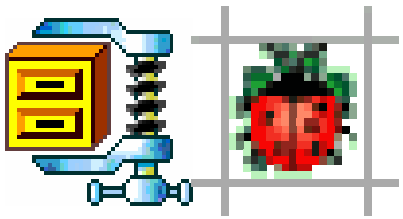
... sind einfach zu verstehen

... lassen sich gut visualisieren

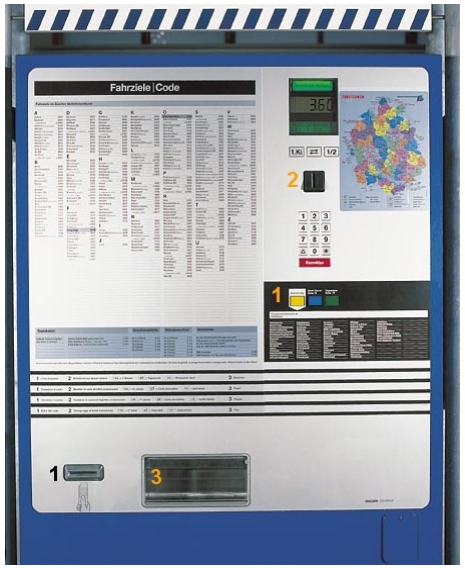
... sind ein mächtiges Berechnungsmodell



Theorie



Automaten im Alltag



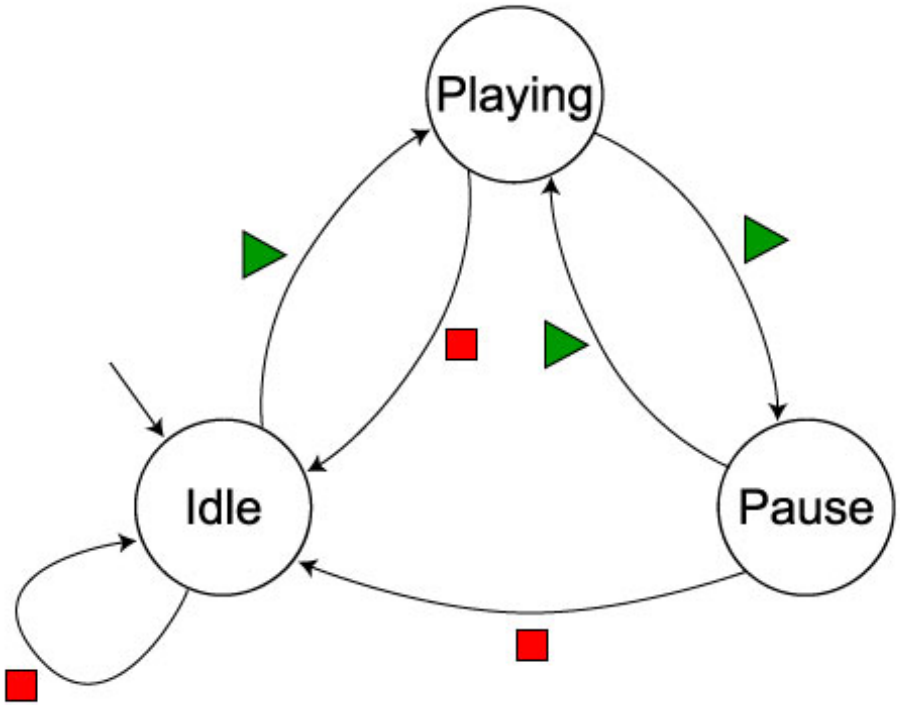
Ein wenig komplexer: Videogerät

- Play / Pause
- Stop

States

Transitions

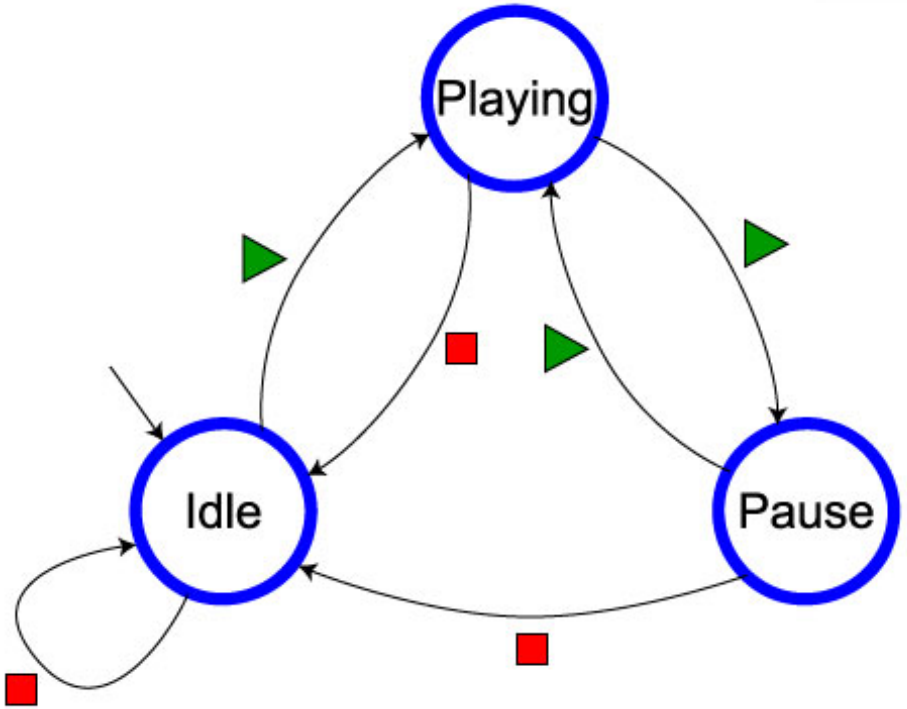
Sensors



Ein wenig komplexer: Videogerät

- Play / Pause
- Stop

- States
- Transitions
- Sensors



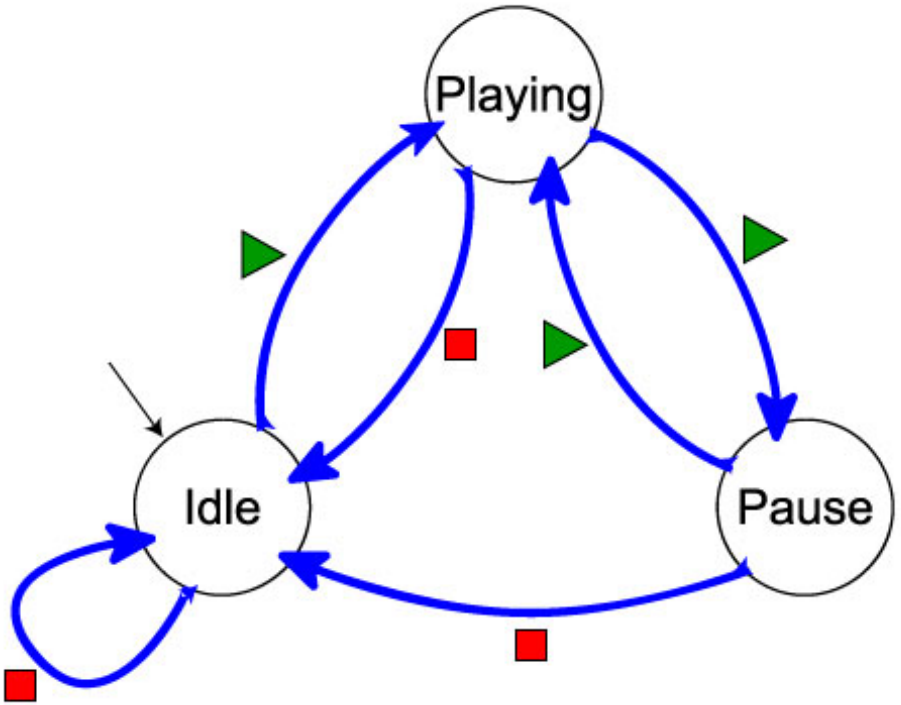
Ein wenig komplexer: Videogerät

- Play / Pause
- Stop

States

Transitions

Sensors



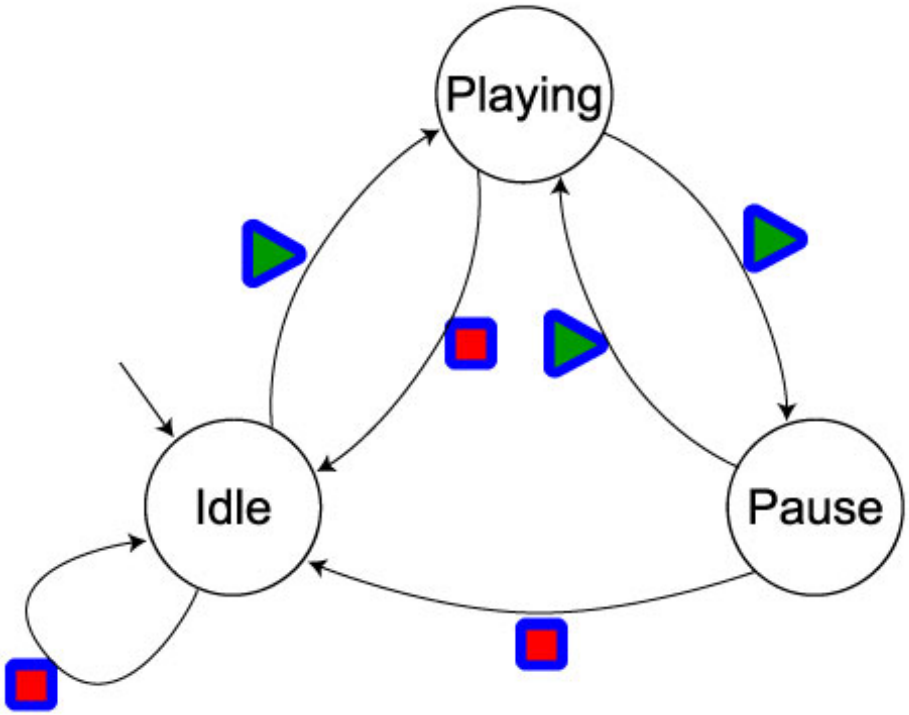
Ein wenig komplexer: Videogerät

- Play / Pause
- Stop

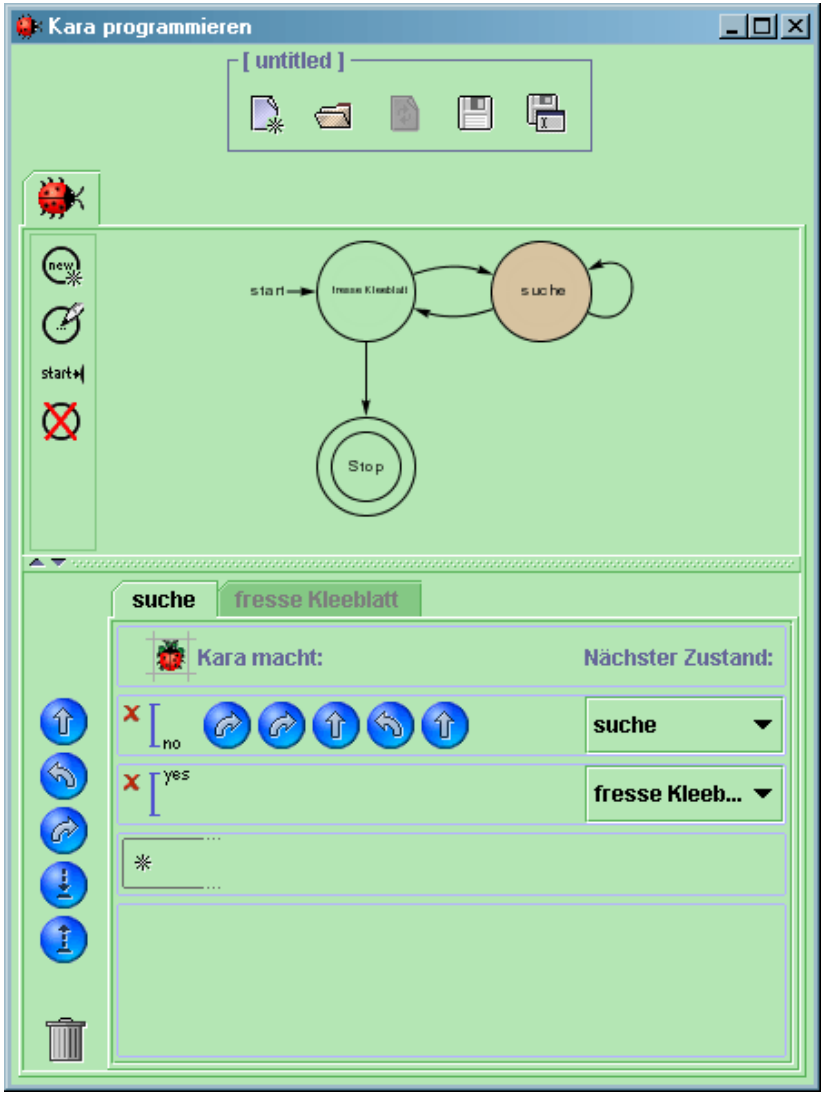
States

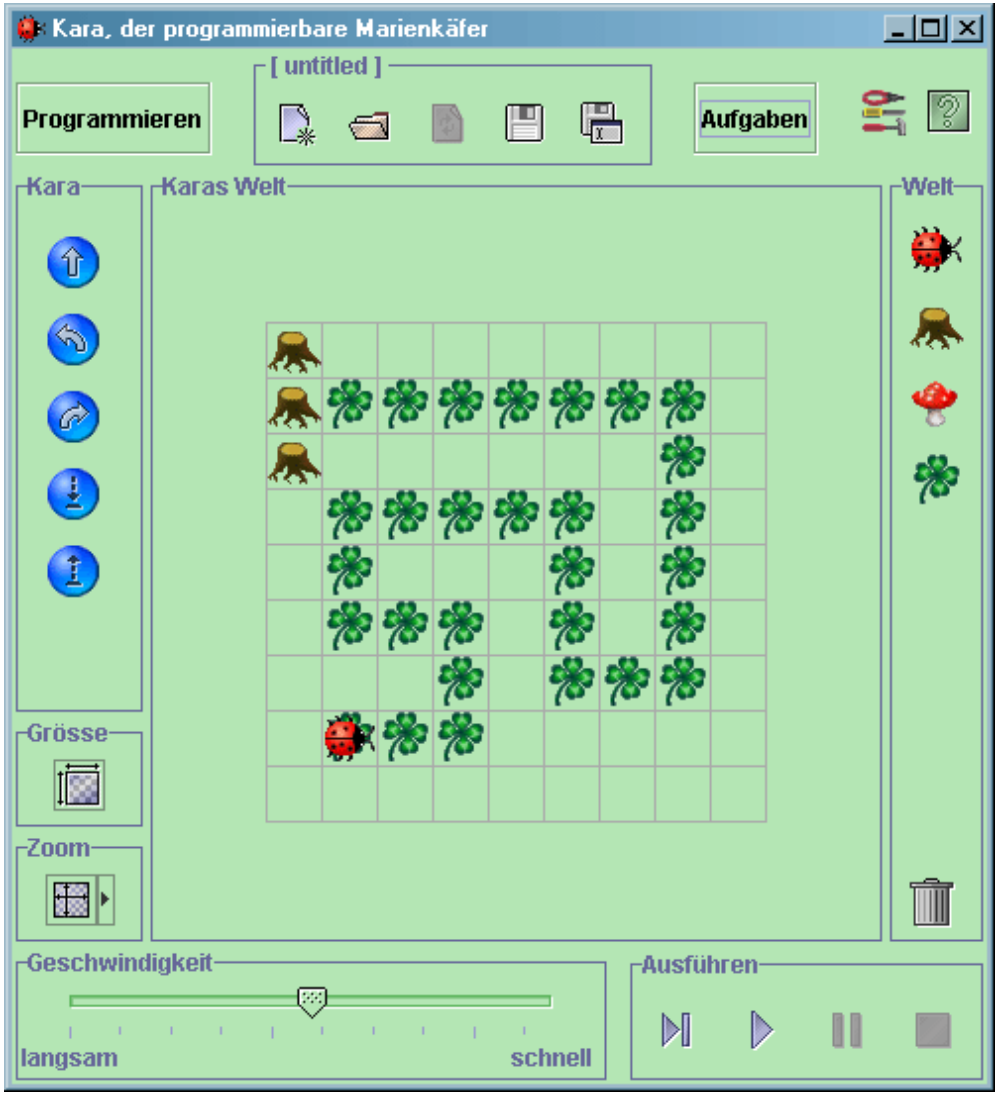
Transitions

Sensors



Die Kara-Umgebung



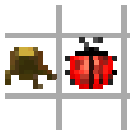


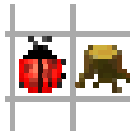
Raimond Reichert - raimond.reichert@acm.org

... und die Käfer-Steuerung

Sensoren:

 Vor Baum?

 Ist links ein Baum?

 Ist rechts ein Baum?

 Vor Pilz?

 Auf Kleeblatt?

Befehle:

 Schritt vorwärts!

 90° Linksdrehung!

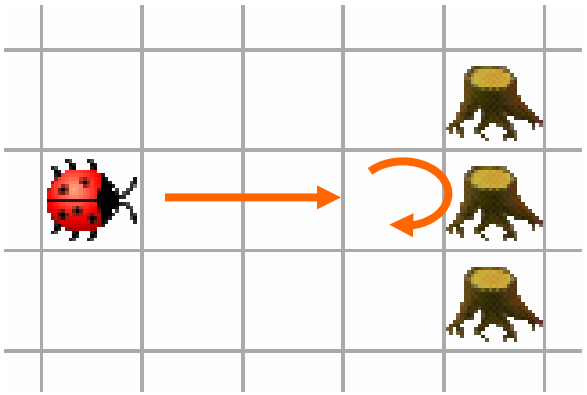
 90° Rechtsdrehung!

 Blatt ablegen!

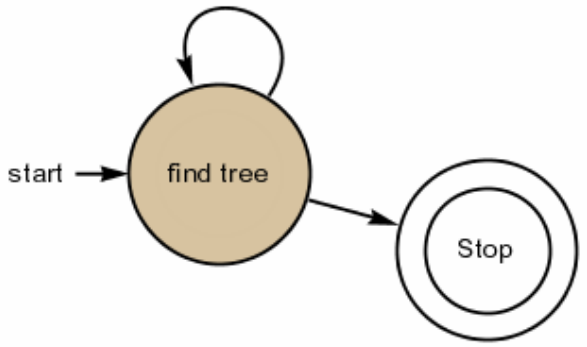
 Blatt aufnehmen!

Raimond Reichert - raimond.reichert@acm.org

Ein „Hello World“ mit Kara



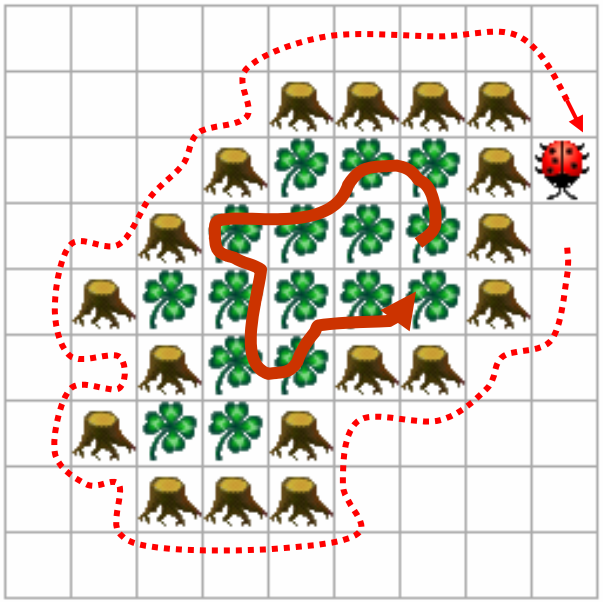
(1) Die Welt und die Aufgabe



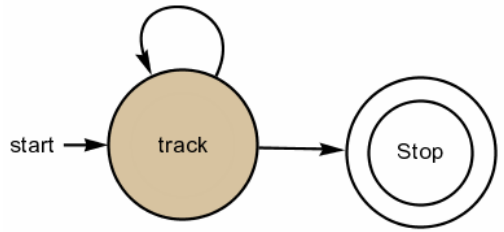
(2) Der Automat

(3) Zustandübergänge für den Zustand „find tree“

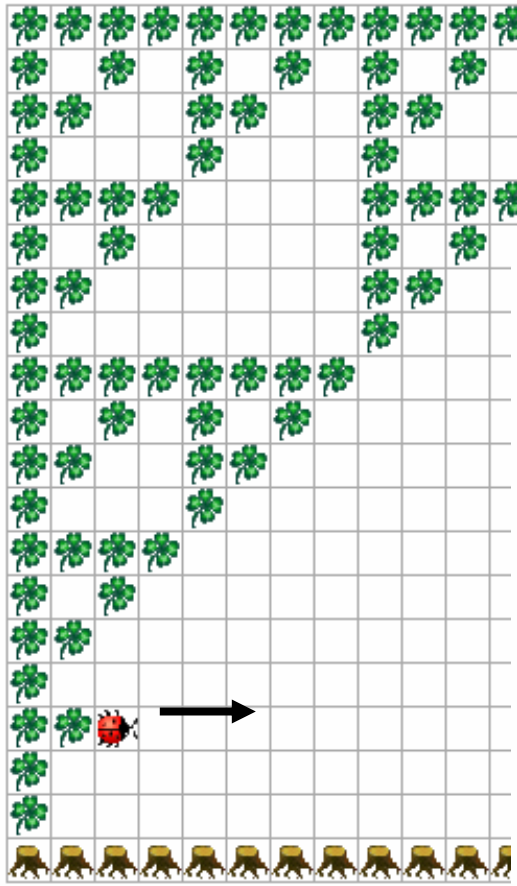
Beispiel mit einem Zustand: Den Bäumen entlang laufen

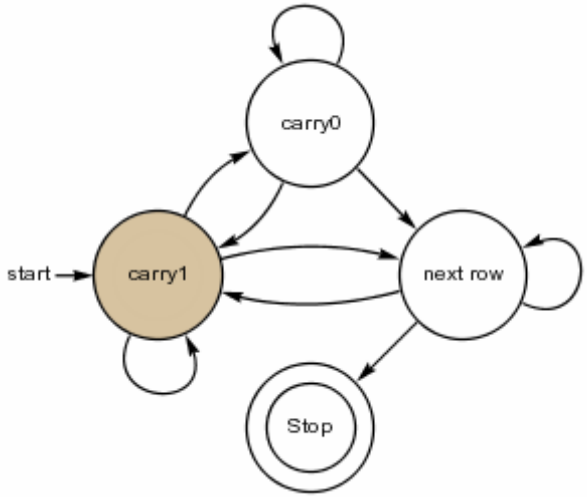


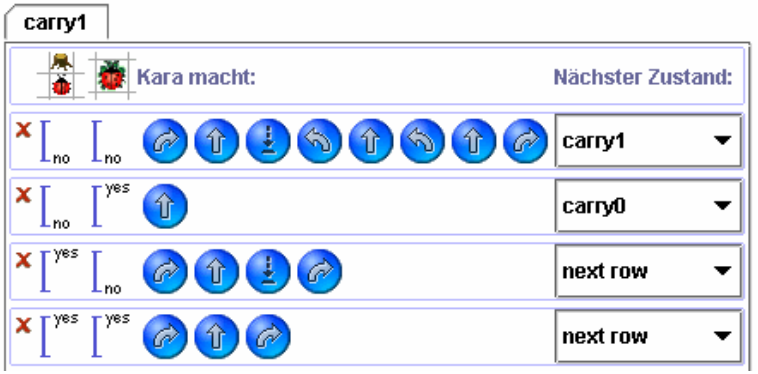
innen oder aussen laufen

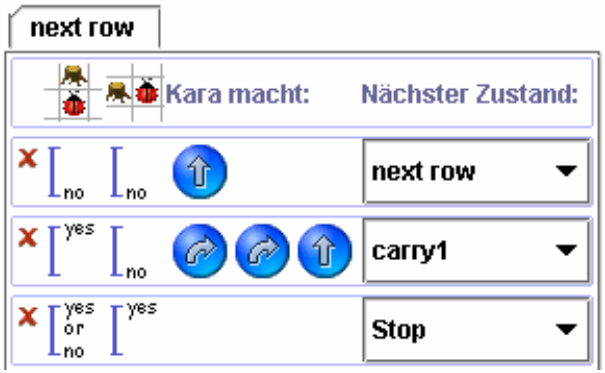


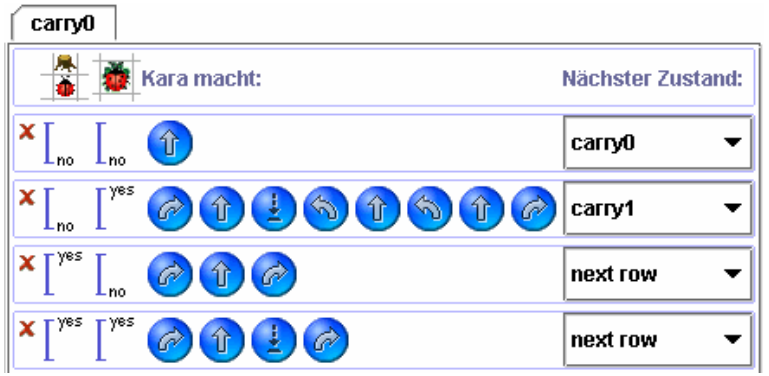
Beispiel mit 3 Zuständen: Kara zeichnet Pascal-Dreieck











Raimond Reichert - raimond.reichert@acm.org

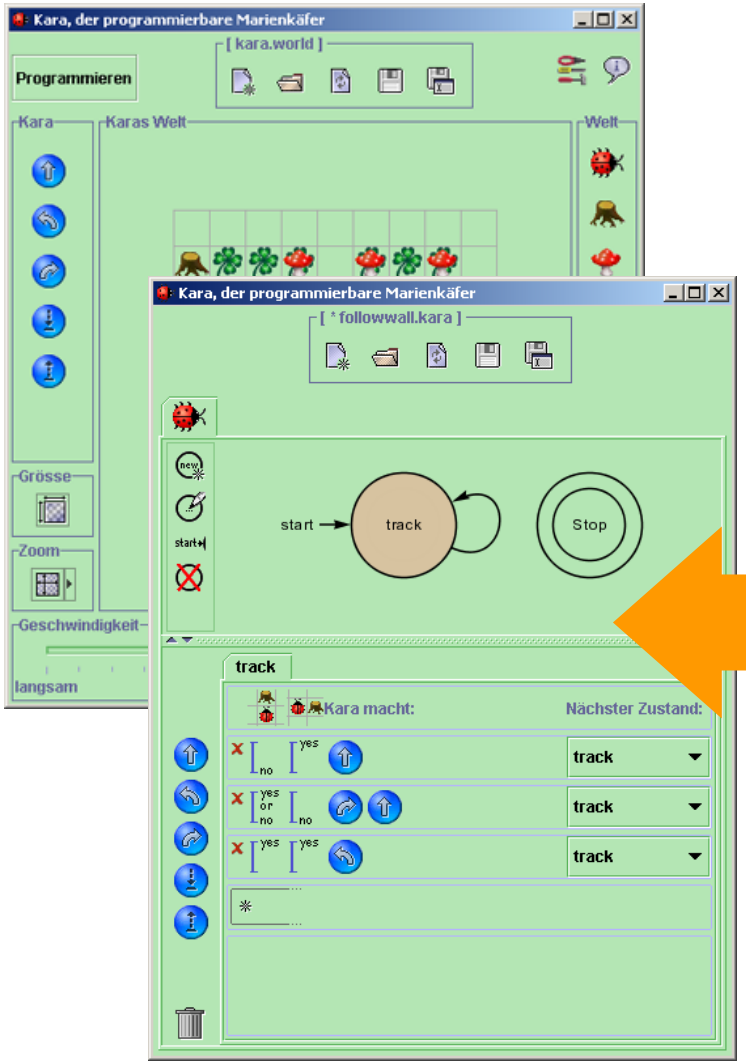
Benefits of Kara in the classroom

- ✓ give students an idea of what is a program
- ✓ allow to pose and solve a wide range of problems in a visual way, with immediate visual feedback
- ✓ emphasise problem solving and Boolean logic
- ✓ allows to give an idea of correctness proofs
- ✓ teach concepts that were valid 20 years ago, and will be valid in another 20 years
- ✓ make programming easy: no textual syntax to learn, only graphical interaction

Kara-Steckbrief

Worum geht‘s	**Einführung in „Programmierdenken“**
Für wen	**Schüler/innen ohne Programmiererfahrung**
Lernziele	• **Grundlagen der Programmierung** • **Modell der endlichen Automaten**
Einsatz-dauer	**6-12 Lektionen**
Technik	Java 1.2, JRE, keine Installation. Windows, Mac OS X, Unix / Linux.

Kara und die Realität: Eine riesige Lücke?



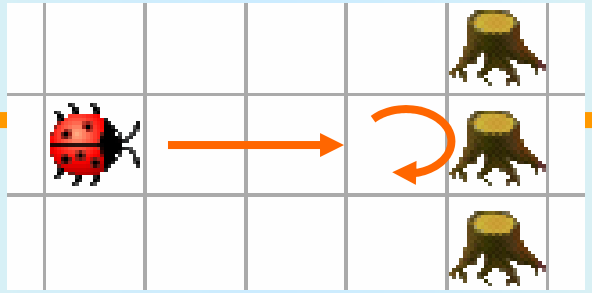
C++, Java, Delphi... professionelle Sprachen

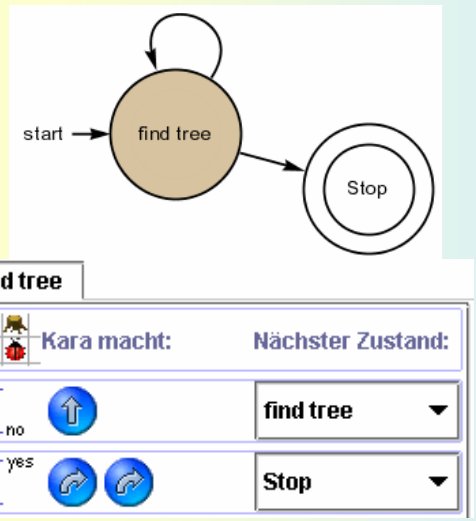
```
private boolean stopInterpreter;

private Runnable interpreterRunnable =
  new Runnable() {
    public void run() {
      try {
        while (!stopInterpreter) {
          if (pauseInterpreter) {
            synchronized (StepableInterpreter.this) {
              while (pauseInterpreter && !stopInterpreter) {
                StepableInterpreter.this.wait();
              }
            }

            se {
            stopInterpreter = !StepableInterpreter.this.step
            listenerSupport.fireStepped();
            try {
              Thread.sleep(waitInterval);
            }
            catch (InterruptedException ie) {
            }
          }
        }
        if (state != STATE_IDLE) {
          stop();
        }
      }
      catch (Exception e) {
        exceptionHandler.handleException(e);
      }
    }
  };
```

Die Lücke überbrücken: Java-, JavaScript-, Python-, RubyKara

Kara: fundamentale Konzepte der Programmierung

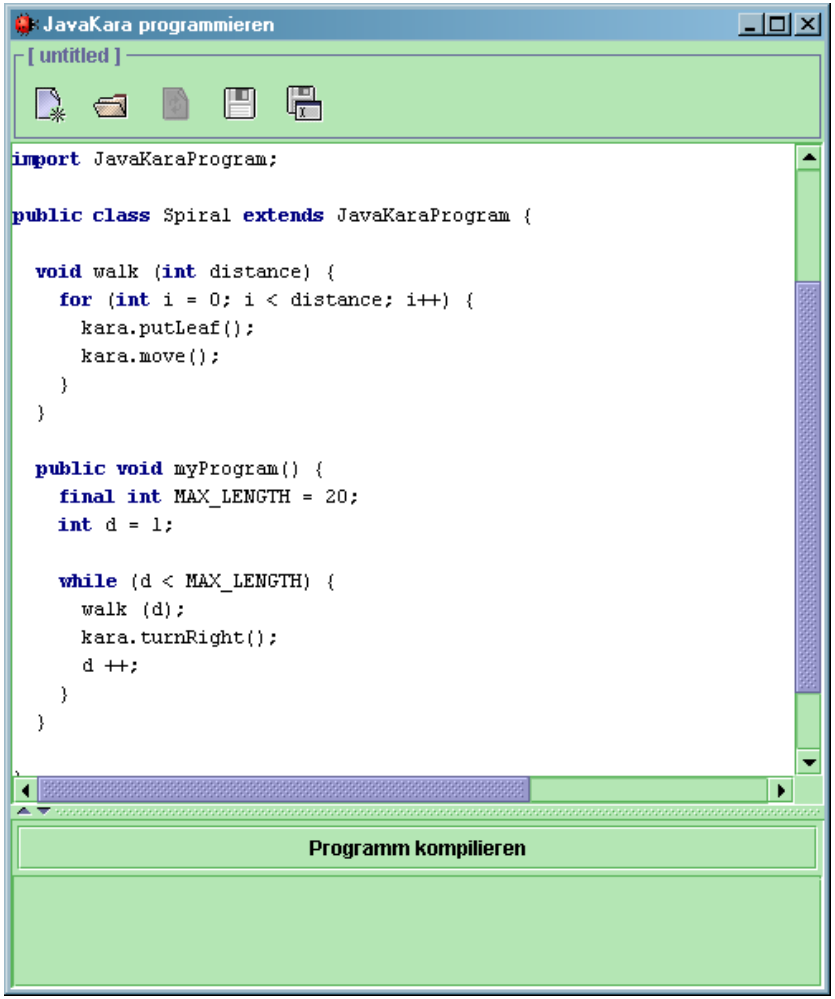
Java etc: aktuelle Programmiersprache

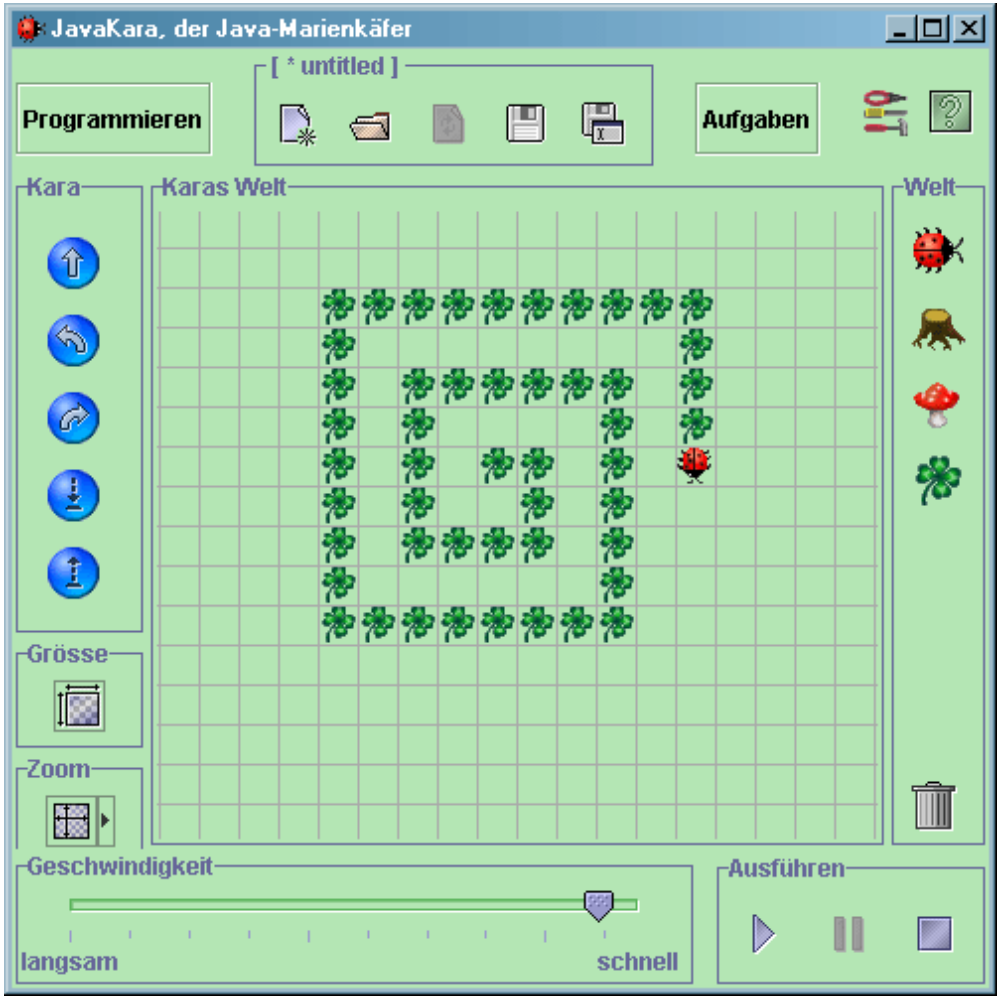





```
public class WalkToTree
 extends JavaKaraProgram {
 protected void myProgram(){
  while (!kara.treeFront()) {
   kara.move();
  }
  kara.turnRight();
  kara.turnRight();
 }
}
```

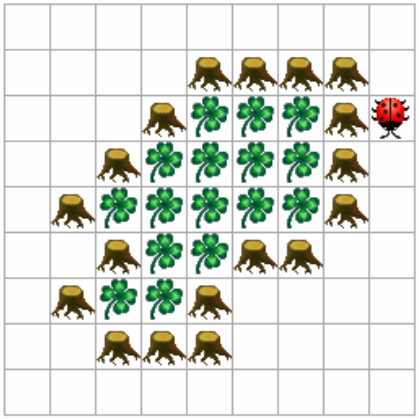
Java-, JavaScript-, Python-, RubyKara





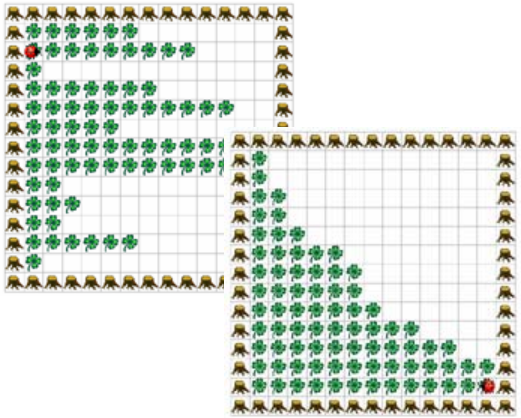
Java-, JavaScript-, Python-, RubyKara programmieren

(1) Kara steuern



```
while (true) {
 if (kara.treeFront() && kara.treeRight()) {
   kara.turnLeft();
 }
 else if (!kara.treeFront()) {
  if (kara.treeRight()) {
   kara.move();
  }
  else {
   kara.turnRight();
   kara.move();
  }
 }
}
```

(2) Zugriff auf Welt

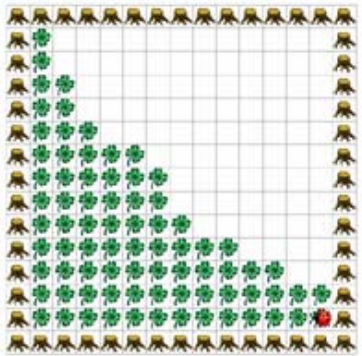


```
int[] numbers = new int[world.getSizeY()];
boolean numbersSorted = false;
permutateNumbers (numbers);
showNumbers (numbers);
do {
 numbersSorted = true;
 for (int i = 0; i < numbers.length-1; i++) {
  if (numbers[i] > numbers[i+1]) {
   swap (numbers, i, i+1); numbersSorted = false;
   showNumbers (numbers);
  }
 }
} while (!numbersSorted);
```

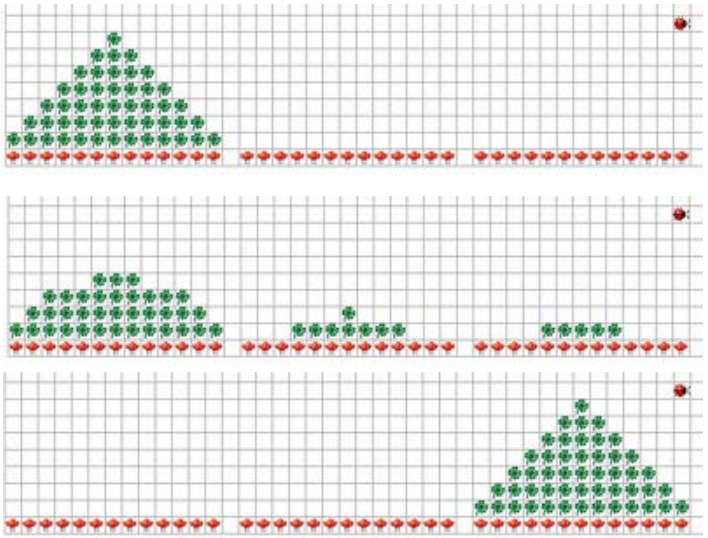
Raimond Reichert - raimond.reichert@acm.org

„Fortgeschrittene“ Java-, JavaScript-, Python-, RubyKara Aufgaben

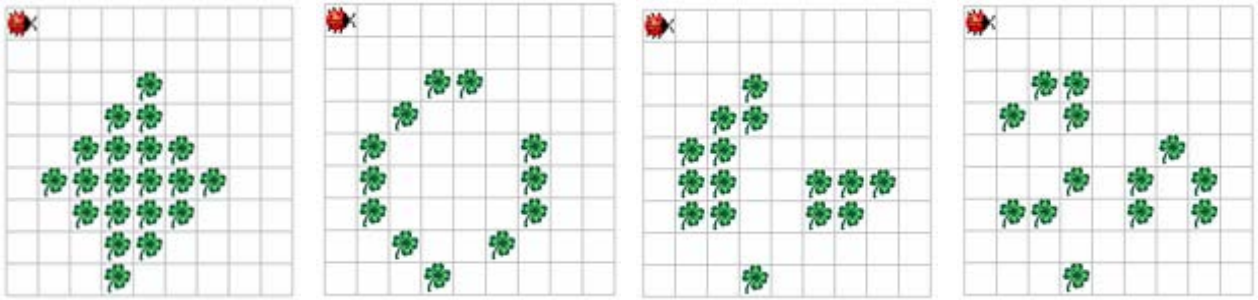
QuickSort



Towers of Hanoi



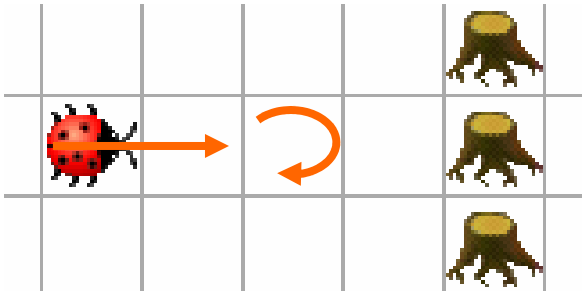
Game of Life



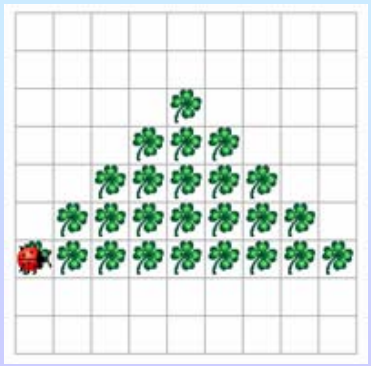
Raimond Reichert - raimond.reichert@acm.org

Von Kara zu Java, JavaScript, Python, Ruby: 1, 2, 3 !

1. Kara
Einführung in Konzepte der Programmierung

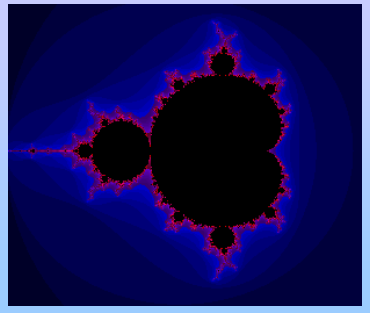


2. Java/...Kara
Einführung in Java-Grundlagen



```
final int HEIGHT = 5;
int currentWidth = 1;
boolean left2right = true;
for (int y = 0; y < HEIGHT-1; y++) {
 paintRow (currentWidth);
 gotoNextRow (left2right);
 left2right = !left2right;
 currentWidth = currentWidth + 2;
}
paintRow (currentWidth);
```

3. Java/...
Forsetzung... so visuell wie möglich



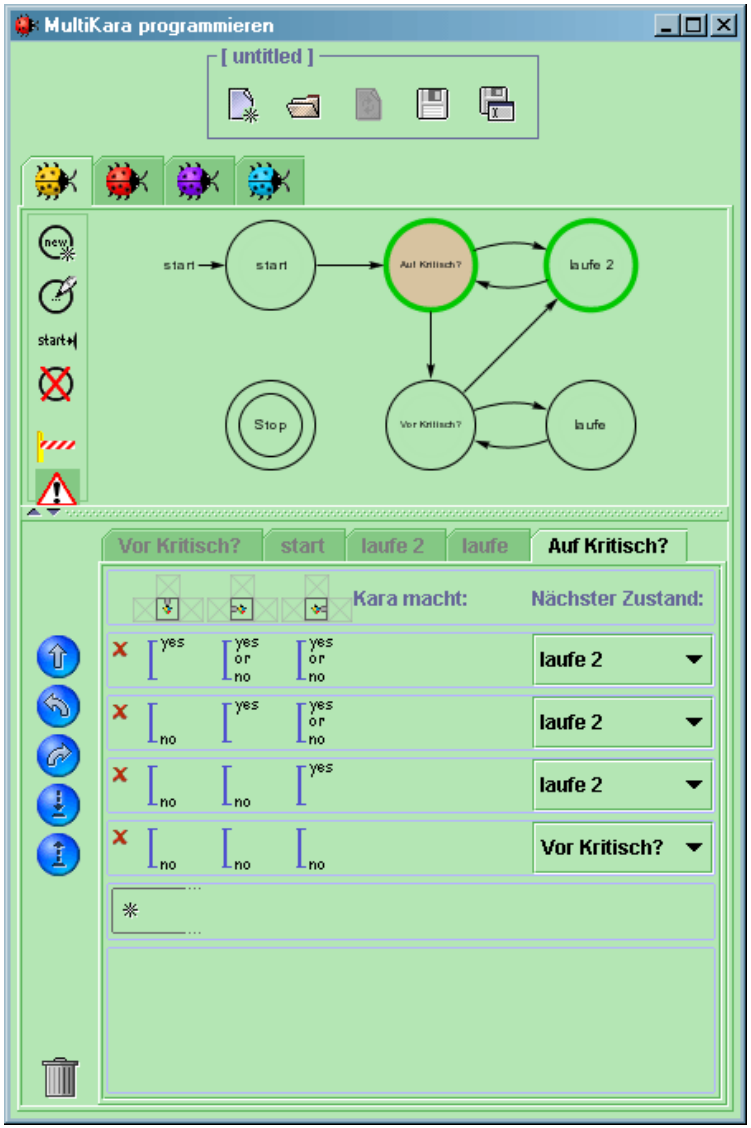
```
float tmpX = 0, tmpY = 0; int i = 0;
do {
 float tmp2X, tmp2Y;
 i++;
 tmp2X = tmpX*tmpX - tmpY*tmpY + x;
 tmp2Y = 2*tmpX*tmpY + y;
 tmpX = tmp2X; tmpY = tmp2Y;
} while (((tmpX*tmpX + tmpY*tmpY) <= 4) &&
(i < 50));
```

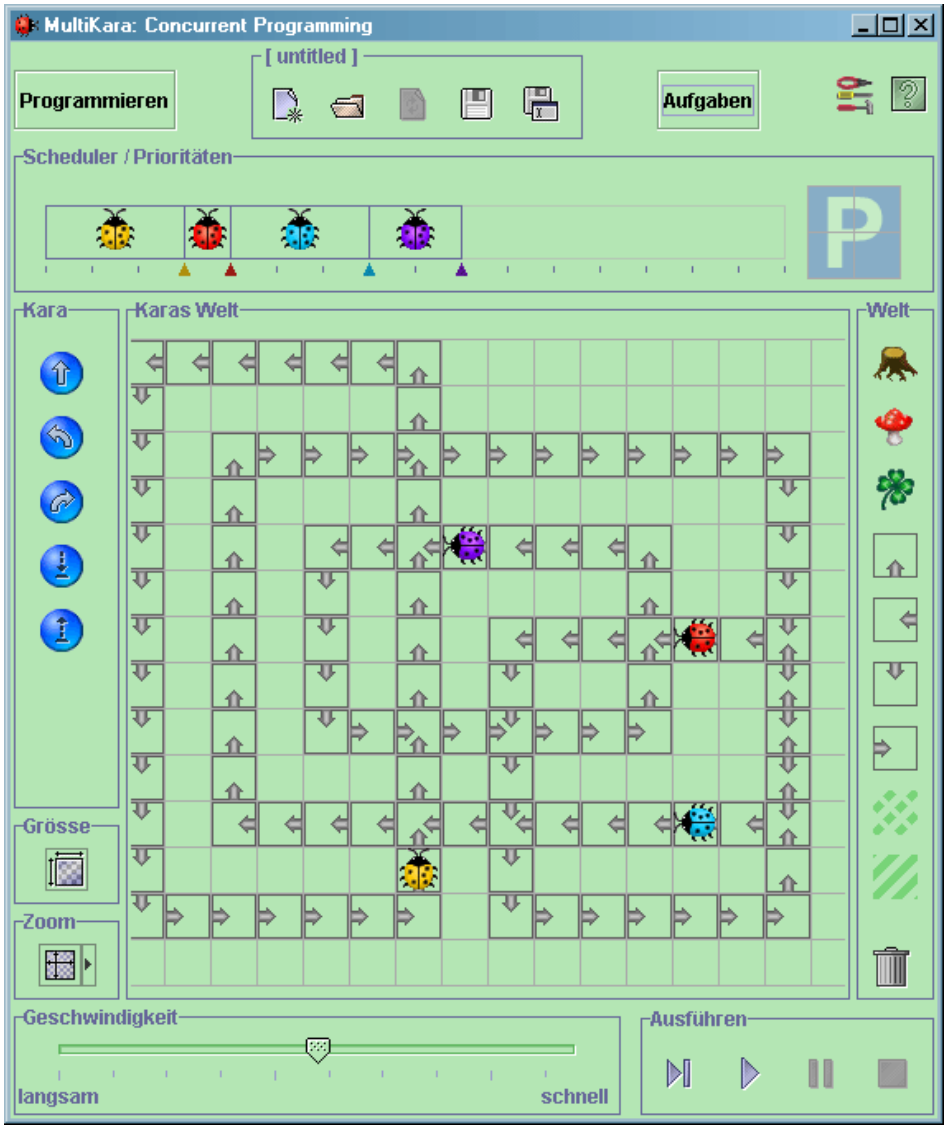
Raimond Reichert - raimond.reichert@acm.org

Java-, JavaScript-, Python-, RubyKara: Steckbrief

Worum geht‘s	**Einführung in die Grundlagen der Sprache (Kontrollstrukturen, Variablen, Methoden ...)**
Für wen	**Schüler/innen ohne Erfahrung mit Sprache (oder verwandter Sprache)**
Lernziele	• **Kontrollstrukturen kennenlernen** • **Eindruck einer professionellen Sprache**
Einsatz-dauer	**10-30 Lektionen**
Technik	Java 6, Java SDK (nur JavaKara), keine Installation. Windows, Mac OS X, Unix / Linux.

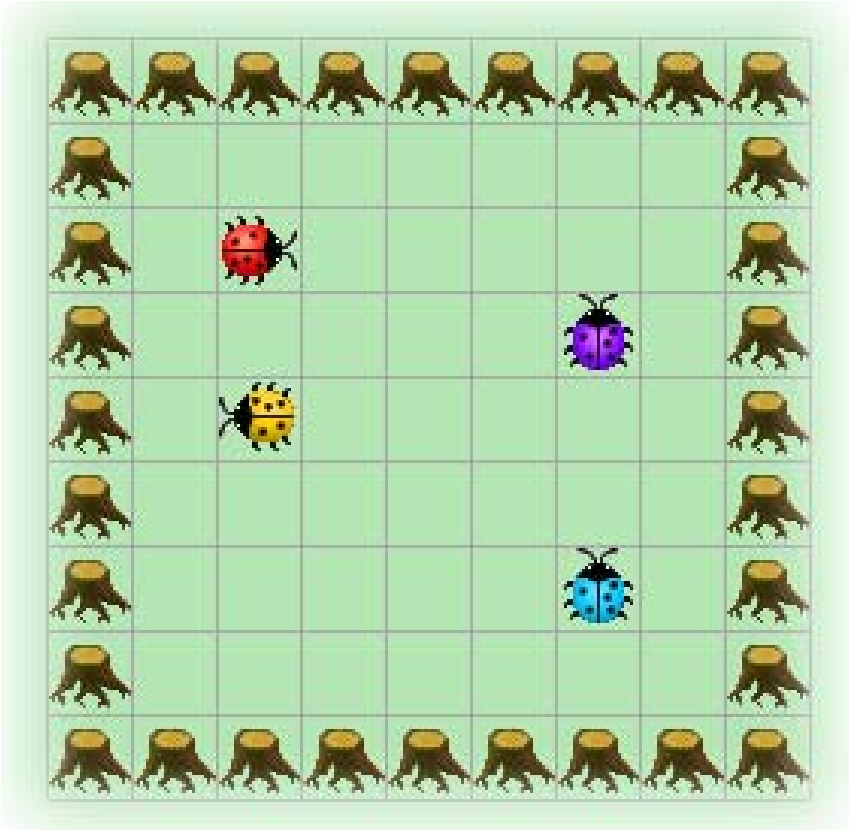
Nebenläufige Programmierung: Die MultiKara-Umgebung





Raimond Reichert - raimond.reichert@acm.org

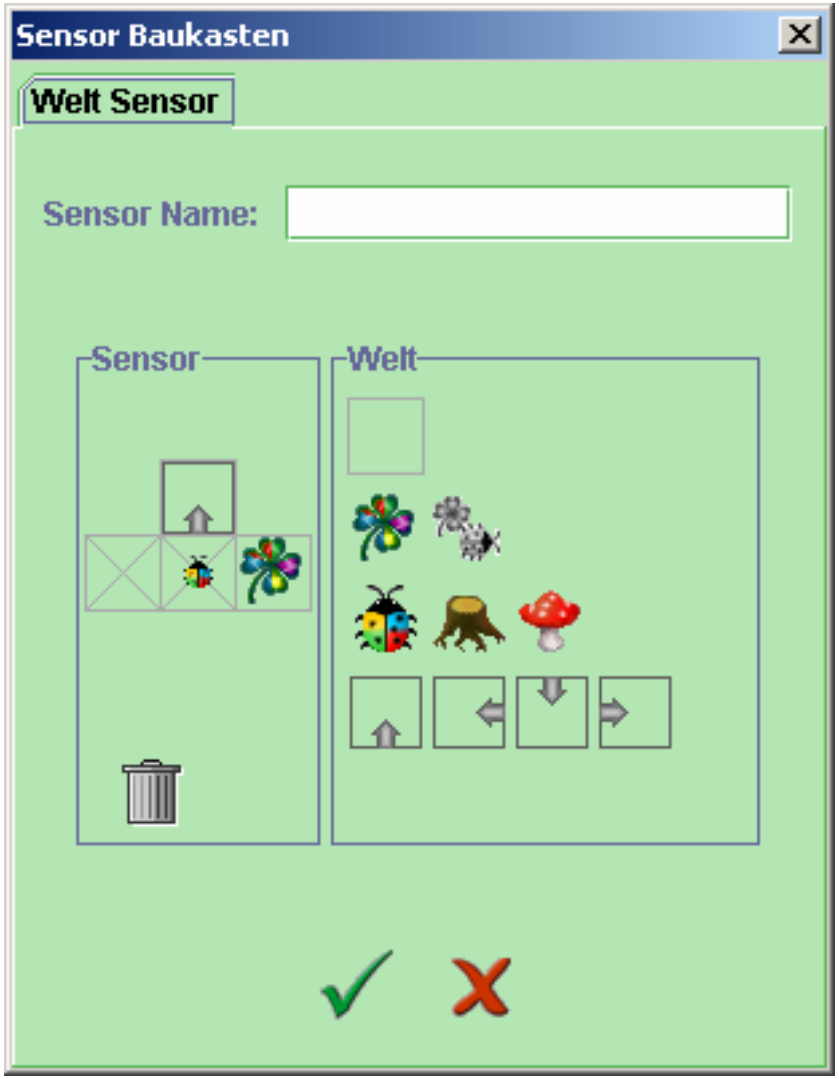
MultiKara oder warum 4 Karas # 4 * 1 Kara



4 Karas in der Welt zulassen und einzeln programmieren.

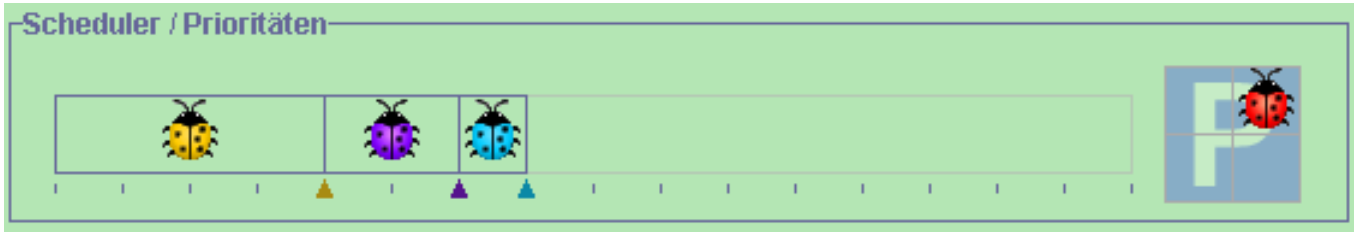
Es scheint so einfach. Oder nicht?

Sensor-Werkzeugkasten



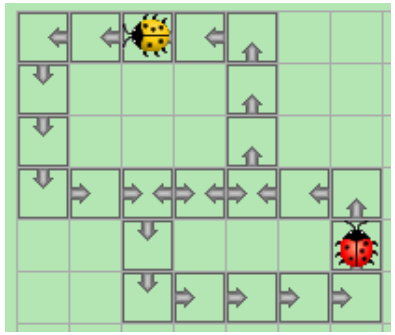
- **Ist ein anderer Kara im Weg?**
- **Ist das Kleeblatt („Markierung“) von mir?**
- **Wo geht die Strasse weiter?**
- **Und das alles vor, links und rechts von mir.....**

Neu in MultiKara: Scheduler und Concurrency-Mechanismen



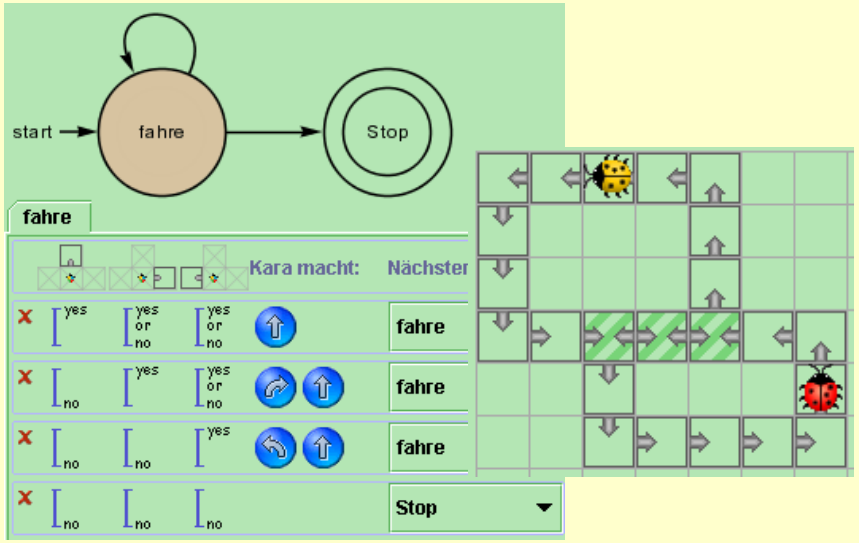
	inclusive "Synchronisation"	exclusive "Gegenseitiger Ausschluss"
world space	**meeting room**	**monitor**
state space	**barrier**	**critical section**

Beispiel für „exclusive“: Gegenseitiger Ausschluss

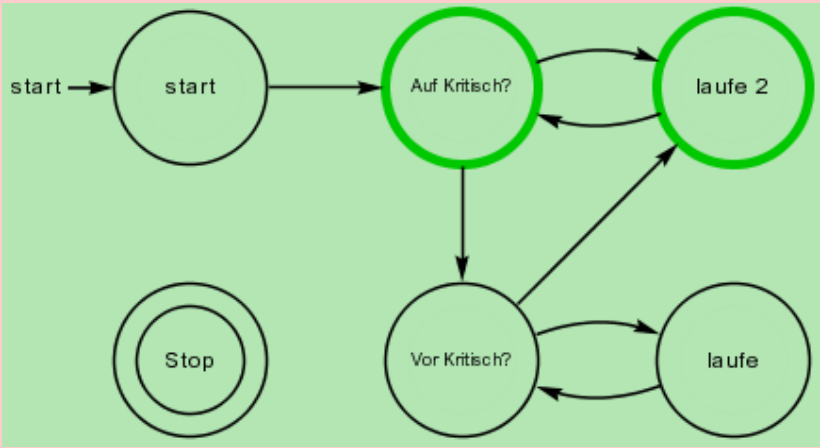


Aufgabe: Endlos Strasse ablaufen, ohne zu kollidieren.

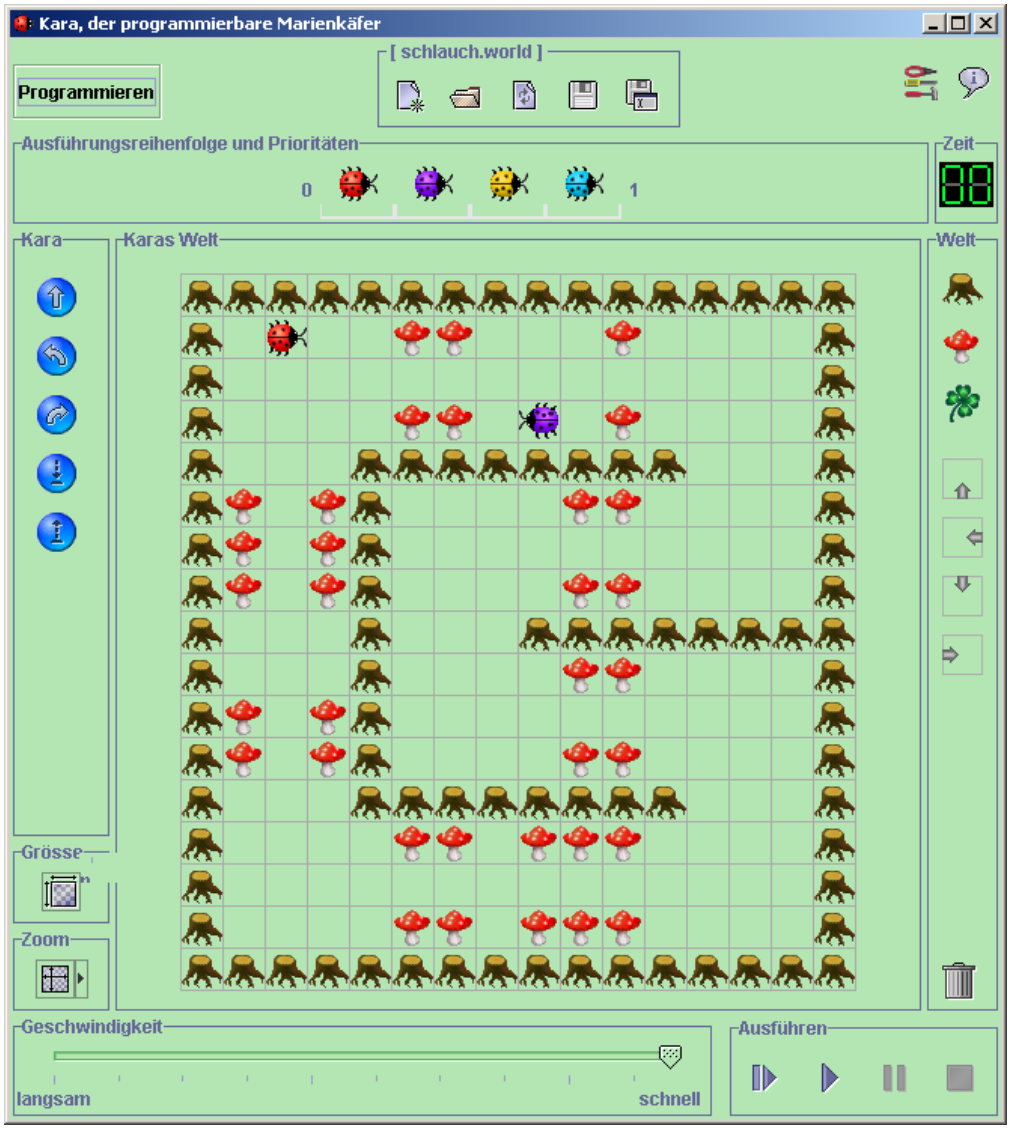
Lösung mit „Monitor“ in der Welt



Lösung mit „Critical Section“ im Automaten

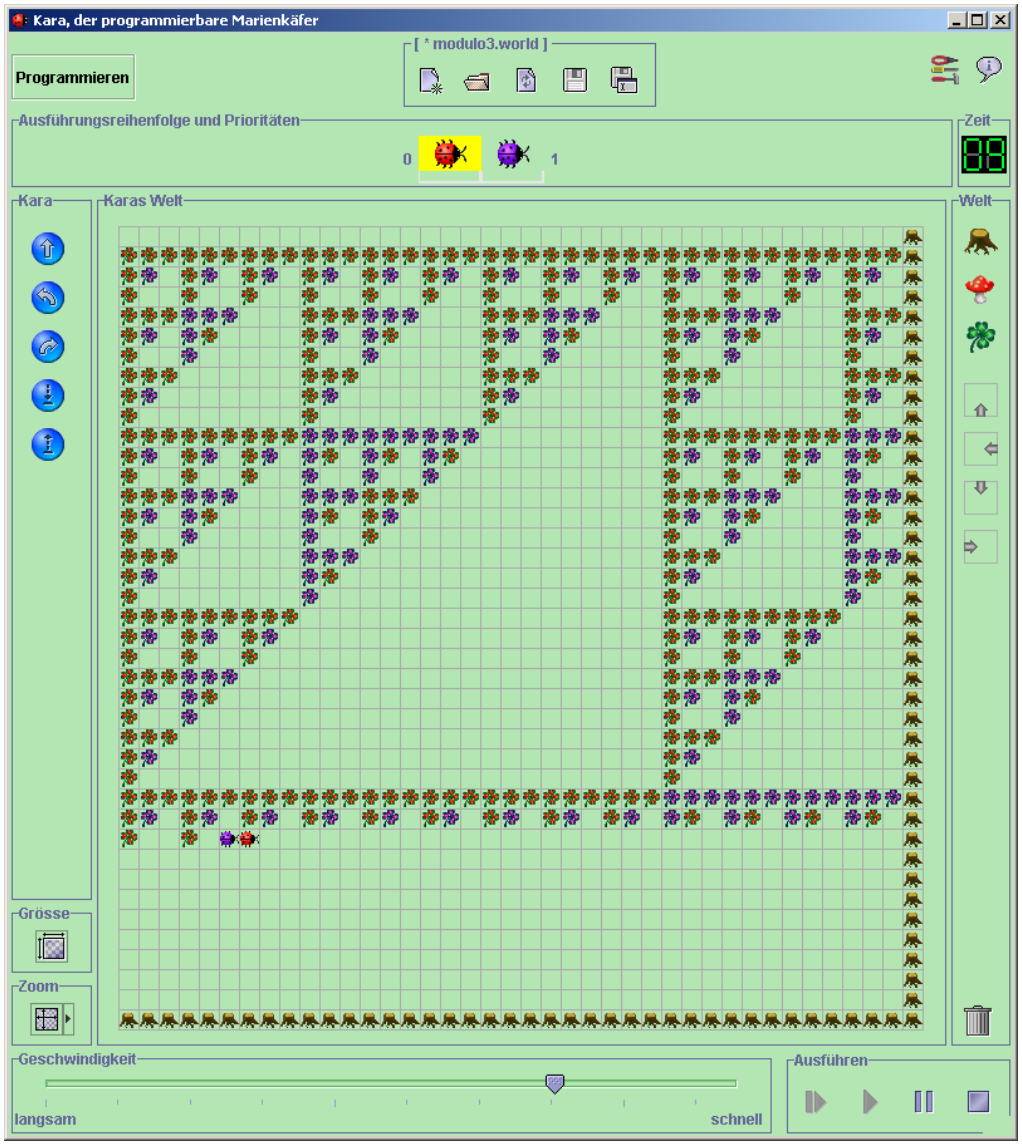


MultiKara examples – distant collision avoidance



Raimond Reichert - raimond.reichert@acm.org

MultiKara examples – cooperation on Pascal modulo 3



Raimond Reichert - raimond.reichert@acm.org

MultiKara – why 4 Karas # 4 * 1 Kara

Example: Fill row with leaves

Ideally, the program is independent of:

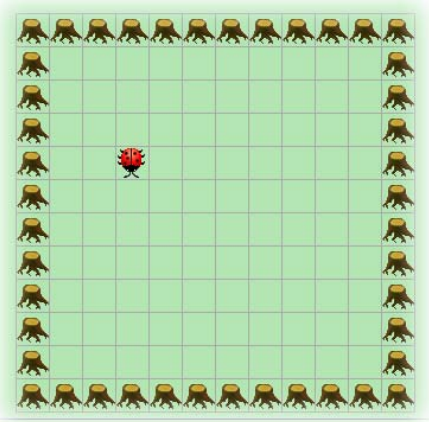
- **length of row**
- **number of Karas**
- **order of execution**
- **priorities**

One solution with two states:

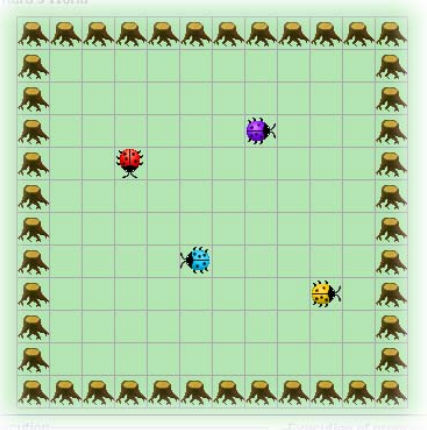
more complex than with one Kara!

MultiKara – why 4 Karas # 4 * 1 Kara

Example: Fill rectangle with leaves



One Kara: no problem.
Walk to corner.
Then fill rectangle.

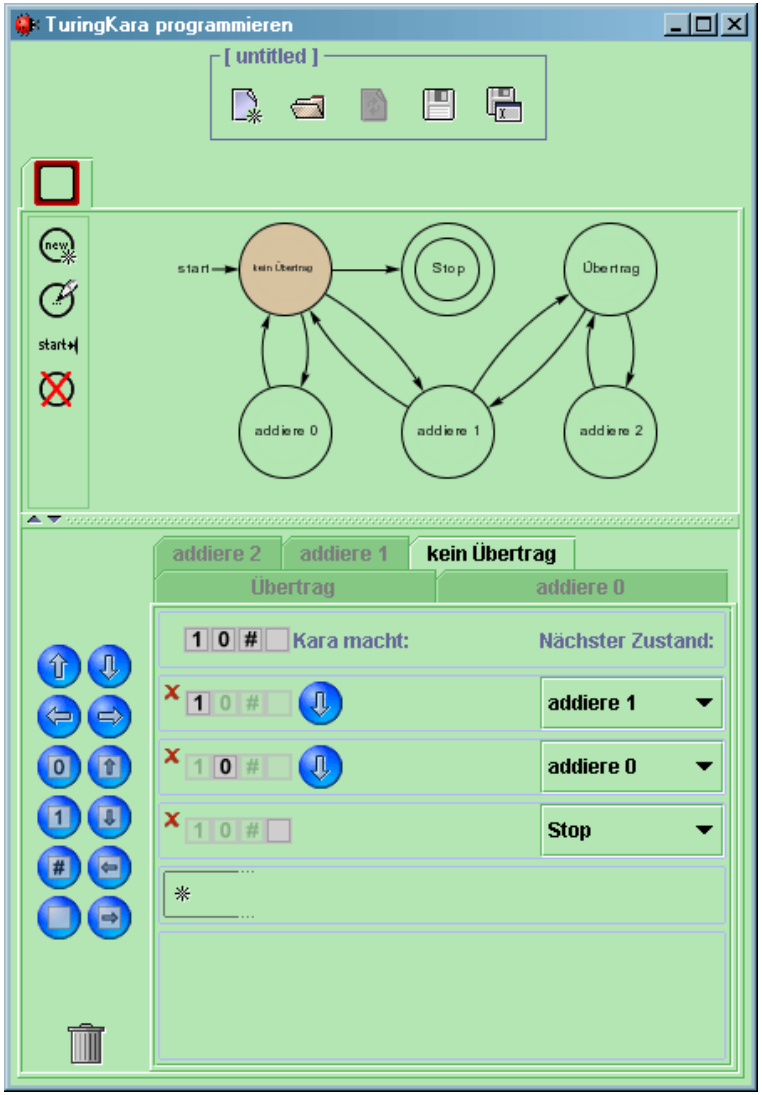


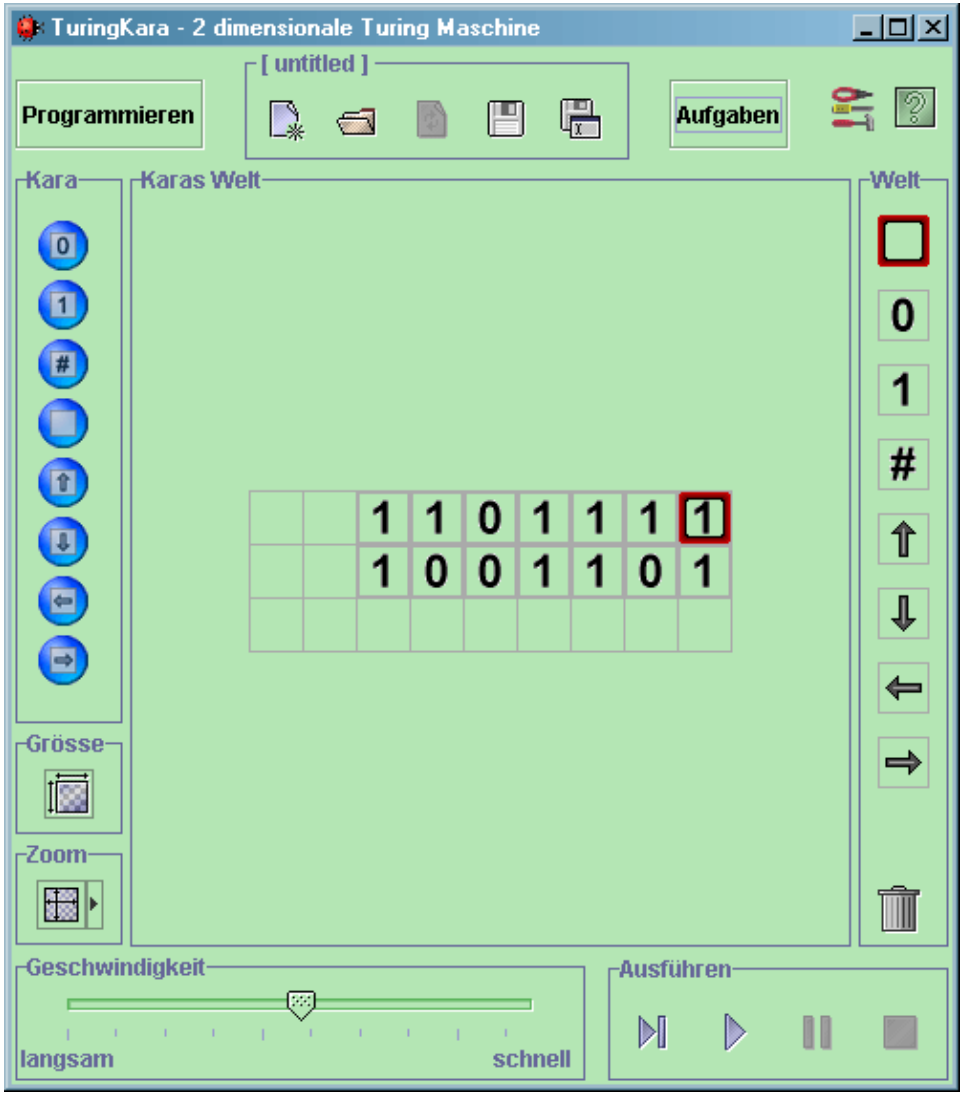
With multiple Karas:
Is it possible with a „reasonable“ number of states?

MultiKara-Steckbrief

Worum geht‘s	**„High-level“ Einführung in nebenläufige Programmierung**
Für wen	**Neulinge in nebenläufiger Programmierung**
Lernziele	• **Unterschiede zu sequentiellen Programmen** • **High-level Concurrency-Mechanismen**
Einsatz-dauer	**6-12 Lektionen**
Technik	Java 1.2, JRE, keine Installation. Windows, Mac OS X, Unix / Linux.

Berechenbarkeit und Turing-Maschinen: Die TuringKara-Umgebung





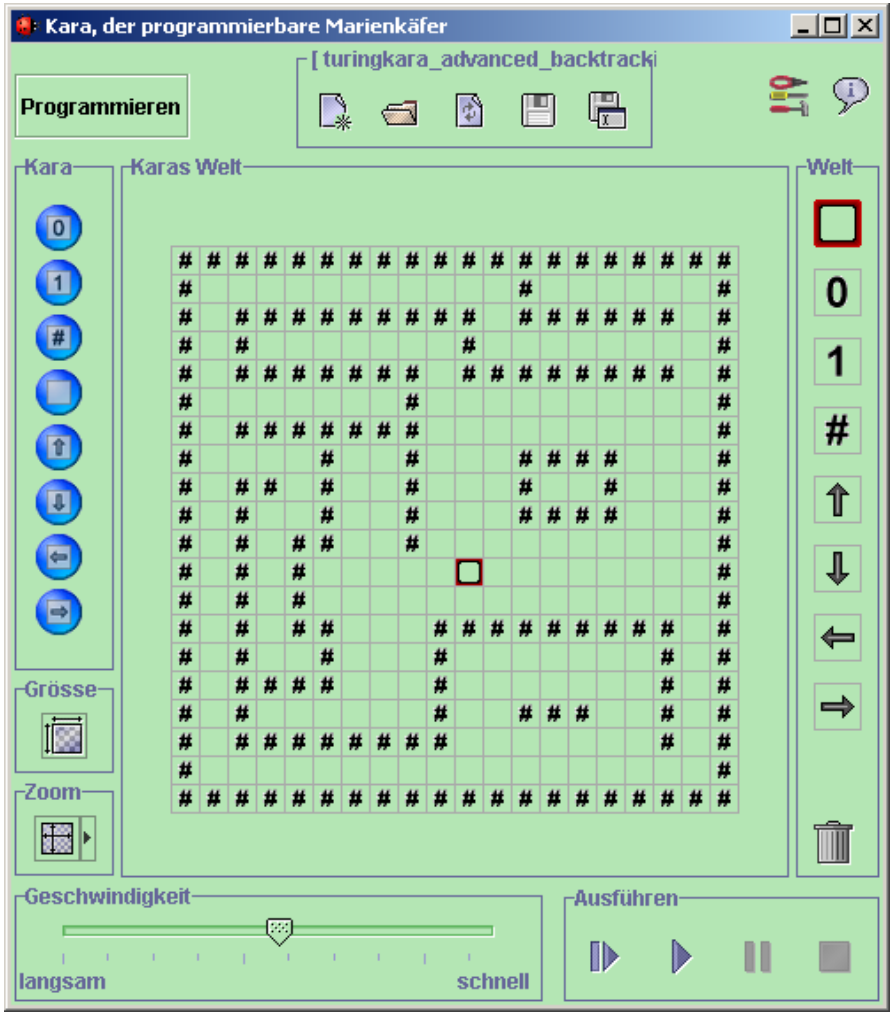
Raimond Reichert - raimond.reichert@acm.org

Vom Blatt zum Band und vom Band zum Blatt

Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child's arithmetic book. In elementary arithmetic the two-dimensional character of the paper is sometimes used. But such a use is always avoidable, and I think that it will be agreed that the two-dimensional character of paper is no essential of computation. I assume then that the computation is carried out on one-dimensional paper, i.e., on a tape divided into squares.

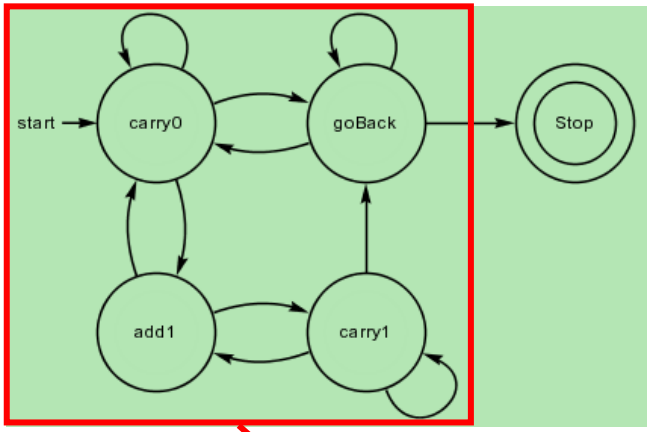
Turing: *On Computable Numbers, with an Application to the Entscheidungsproblem.*
Proceedings of the London Mathematical Society, 1936-1937.

Die Welt von TuringKara



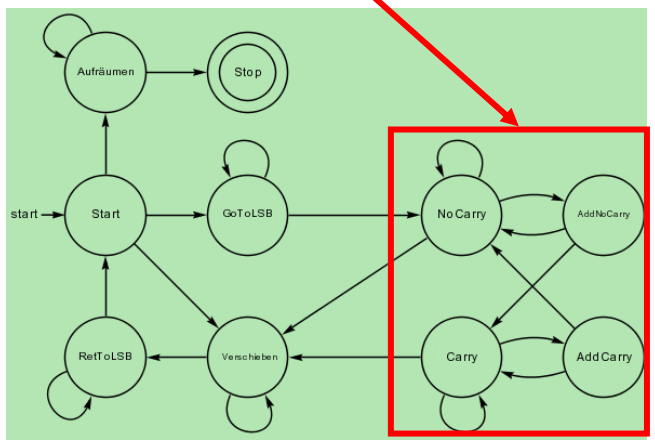
- **Alphabet**
  0, 1, #, , ←, ↑, →, ↓
- **Lese-/Schreibkopf in vier Richtung bewegbar**
- **Lese-/Schreibkopf kann jedes Feld betreten**
- **Schreibbefehle über-schreiben Inhalt des aktuellen Feldes**

TuringKara – oder wie man einem Computer das Rechnen beibringt



#	0	1	0	1	1	1	0	#
#	0	0	0	0	1	0	1	#
#	0	0	1	0	0	0	1	#
#	0	0	0	0	0	1	1	#
#								#

Addition...



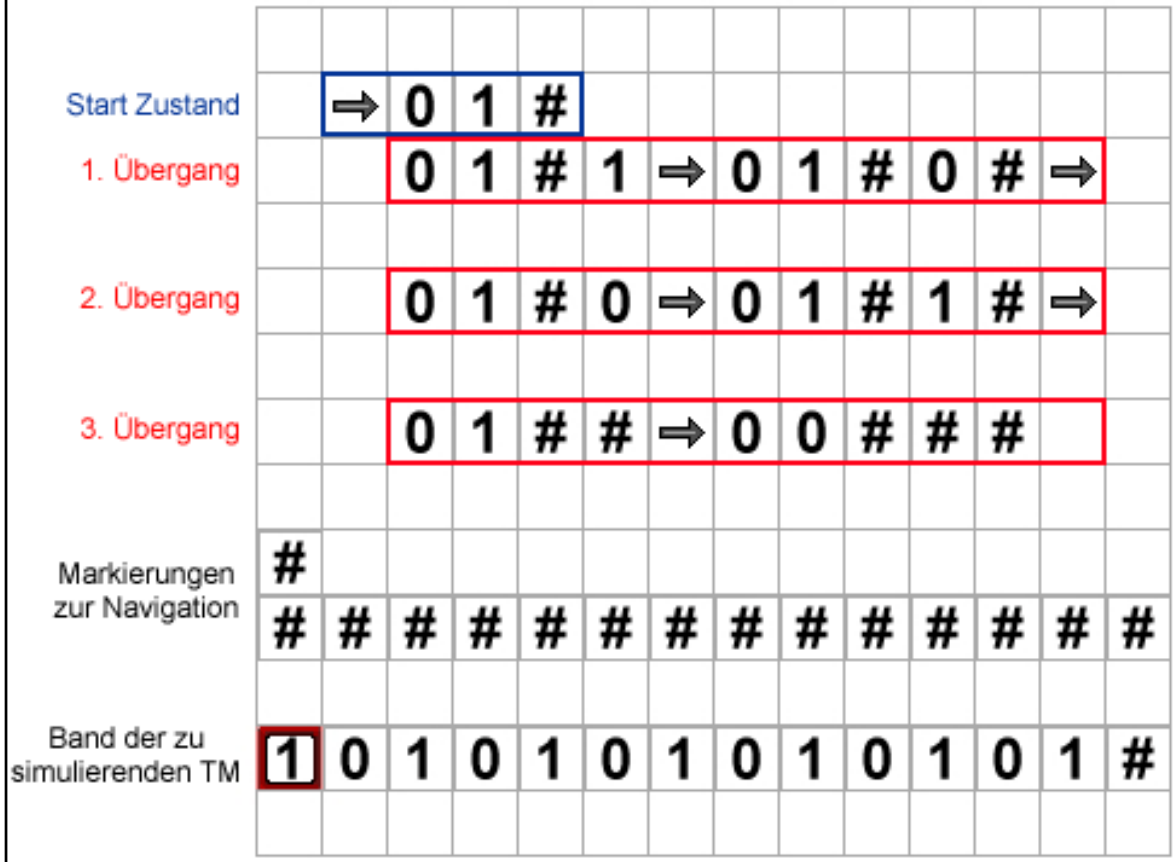
1	0	0	1	#	1	0	1
0	0	0	0	0	0	0	0

... Multiplikation

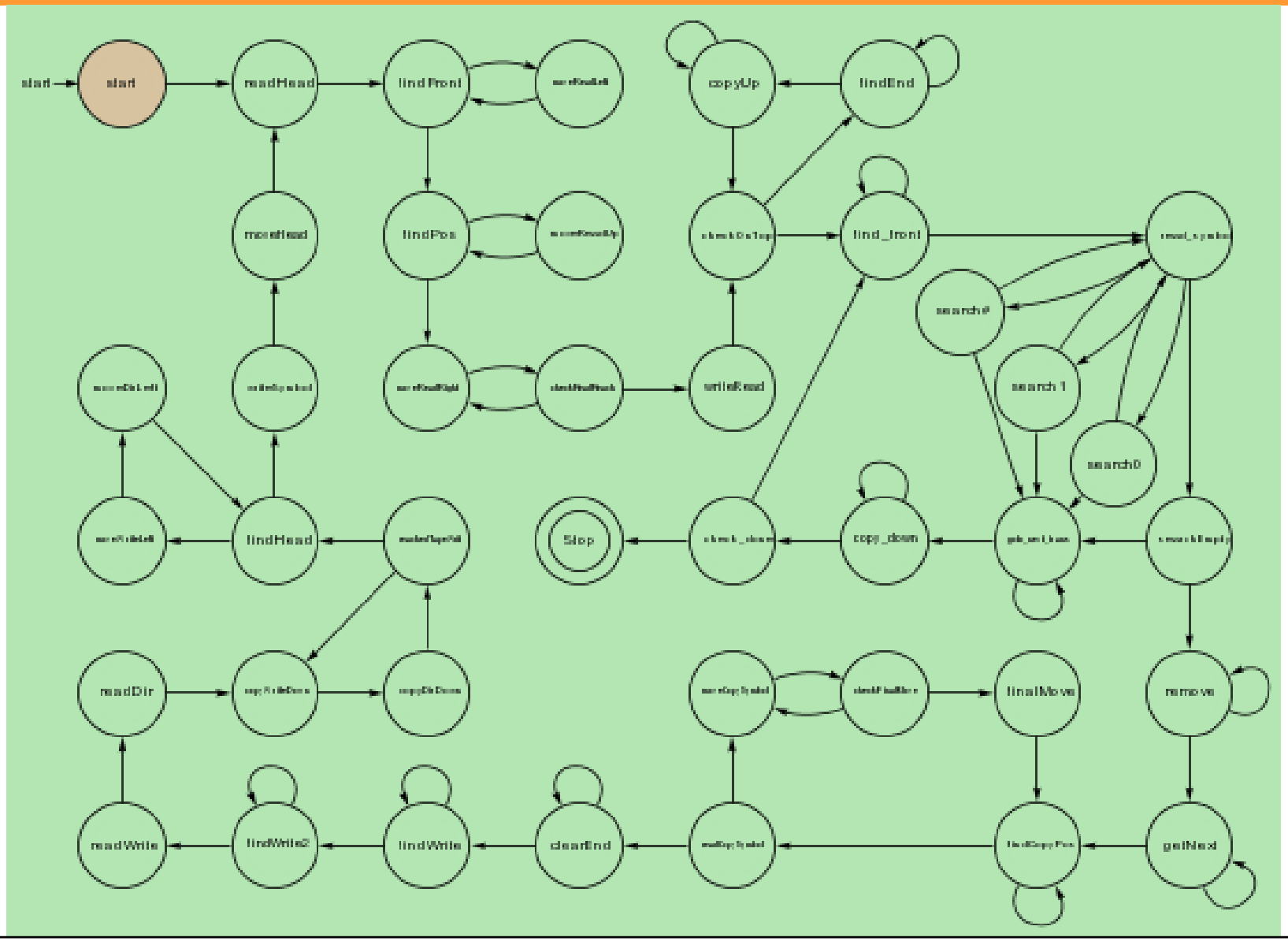
... und vieles mehr!

Das „ultimative Beispiel“: Die Universelle Turing-Maschine

Die TuringKara-UTM simuliert beliebige Standard-Turing-Maschinen.



Die TuringKara-UTM mit 41 Zuständen ...

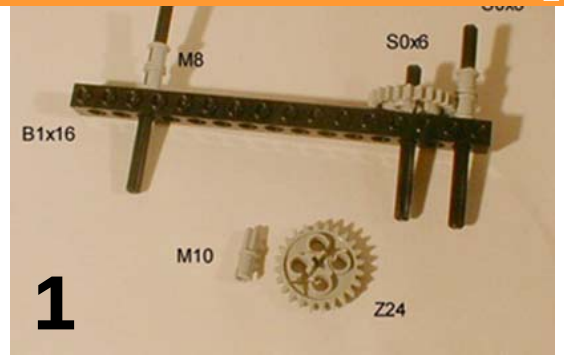


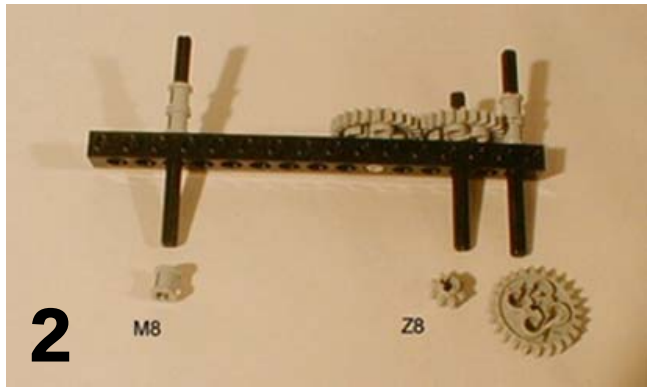
TuringKara-Steckbrief

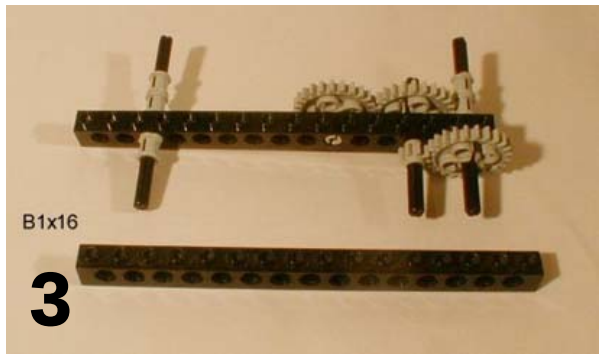
Worum geht‘s	**Einführung Berechenbarkeit am Beispiel von Turing Maschinen**
Für wen	**Unterricht zur theoretischen Informatik**
Lernziele	• **Berechenbarkeit: Modell-Definition** • **Universell: Turing-Maschinen ≅ PC**
Einsatz-dauer	**6-12 Lektionen**
Technik	Java 1.2, JRE, keine Installation. Windows, Mac OS X, Unix / Linux.

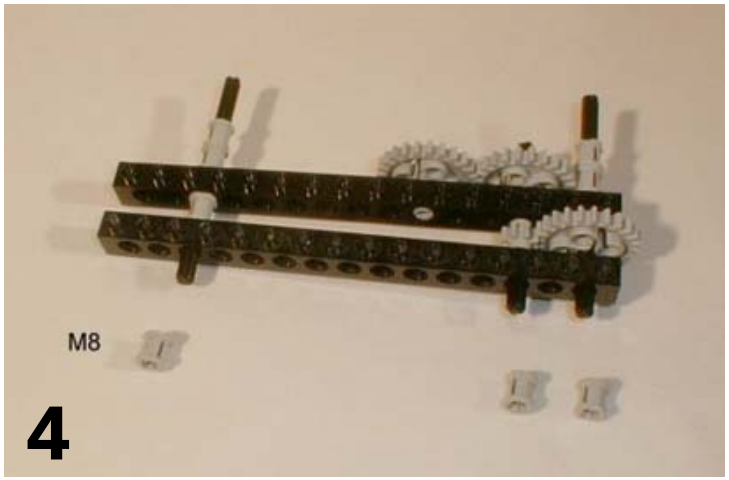
LegoKara: a physical prototype

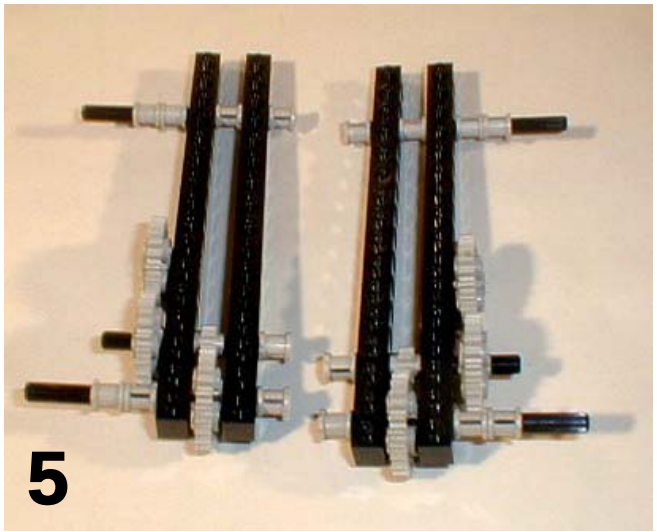
LegoKara: Bastelanleitung (Schritt 1/6)





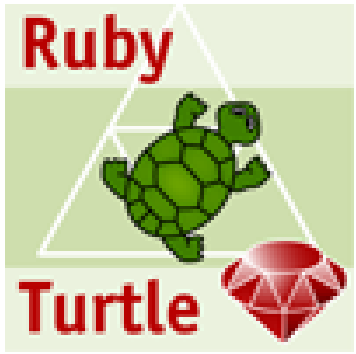


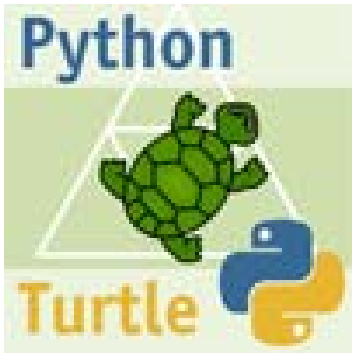




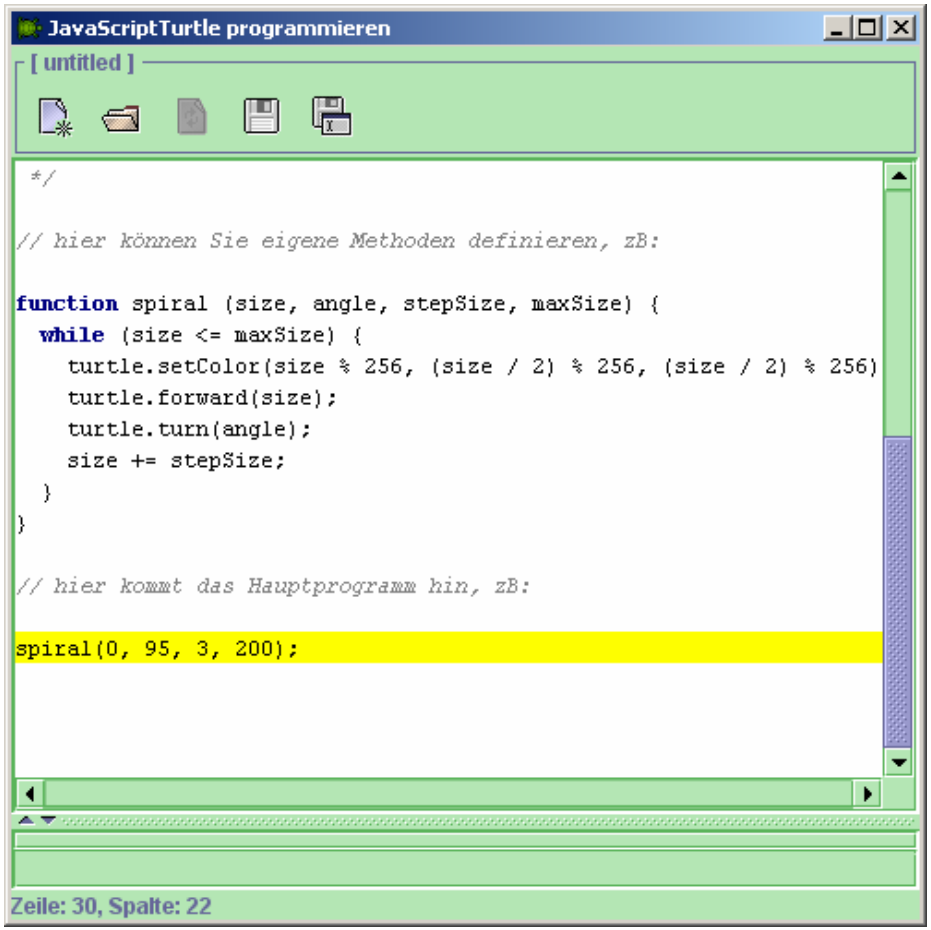
Inhalt

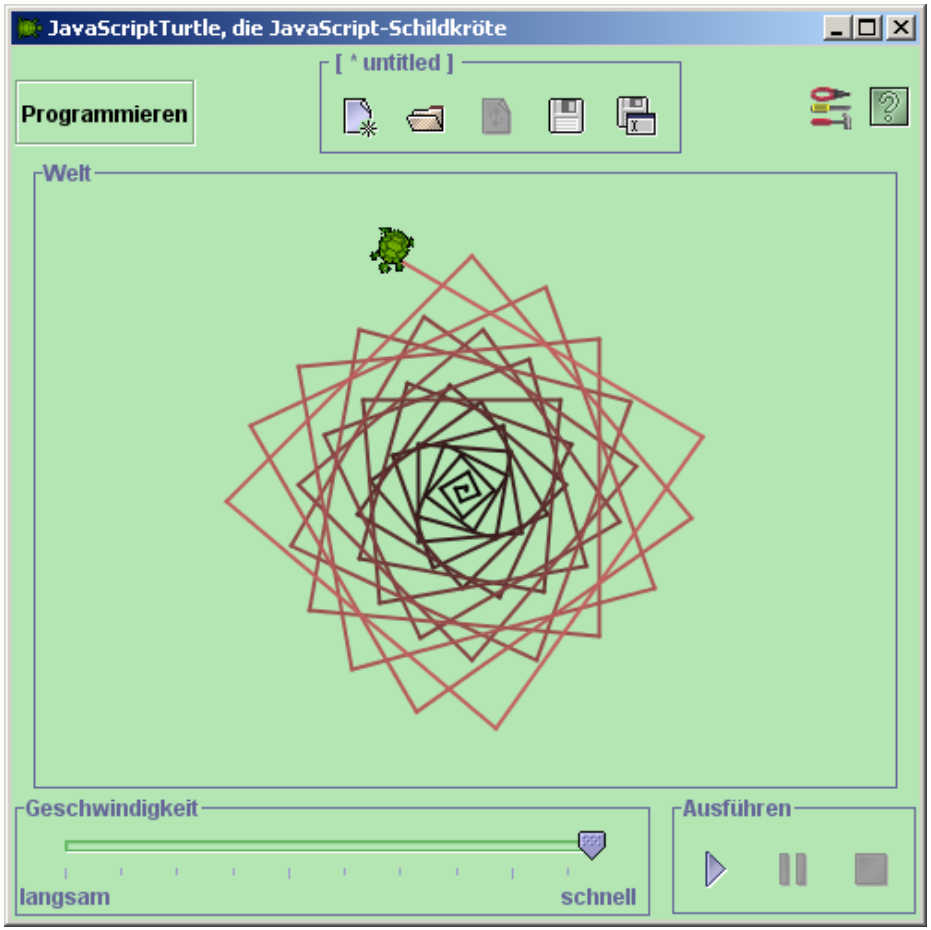
Lernumgebungen fürs Programmieren: Turtles... die gute alte Schildkröte





Java-, JavaScript-, Python-, RubyTurtle





Raimond Reichert - raimond.reichert@acm.org

Inhalt

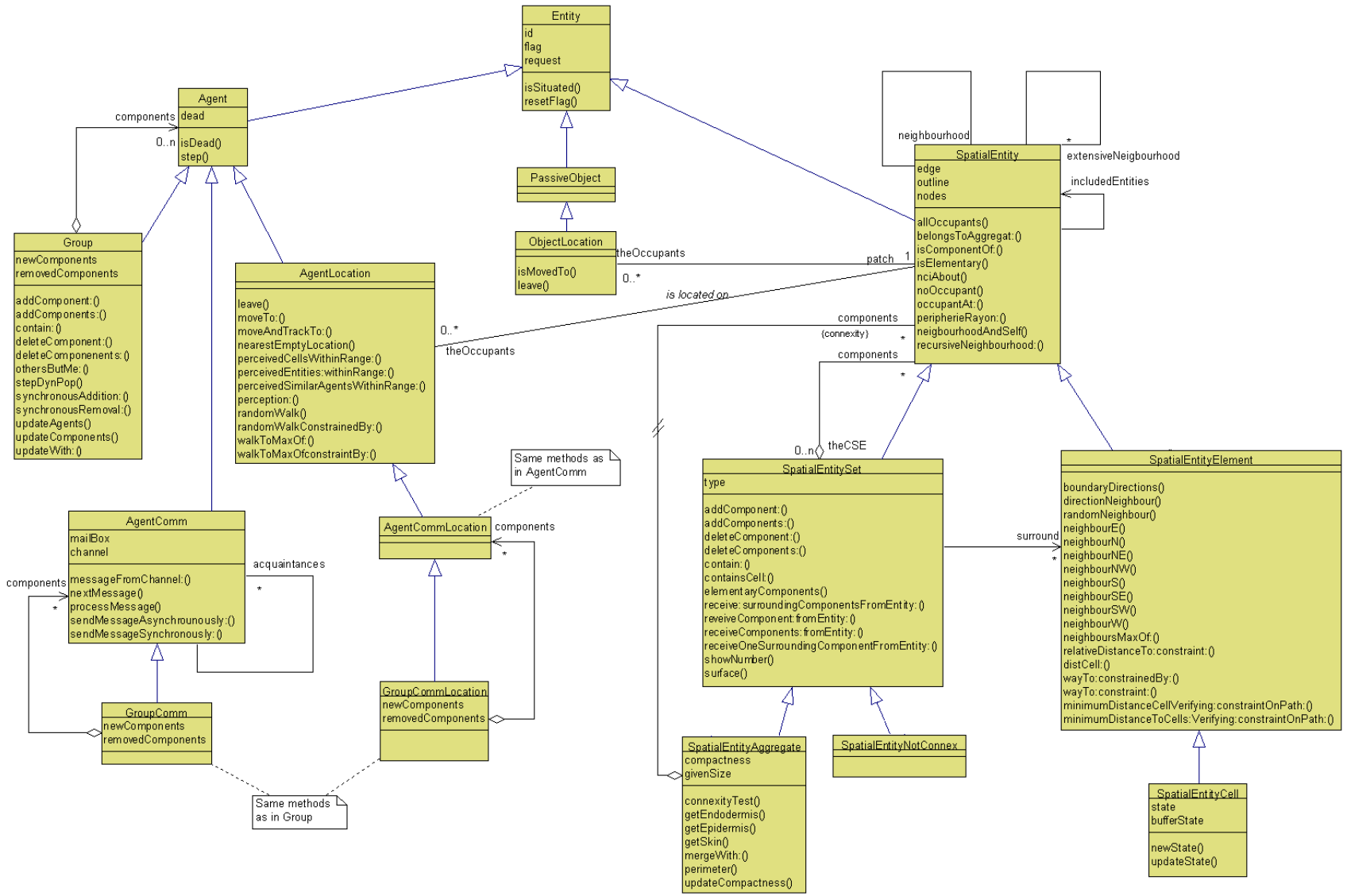
Komplexe Systeme im Informatikunterricht – eine didaktische Herausforderung

Die Informatik stellt Methoden bereit, um grosse und komplexe Probleme zu lösen.

Diese Methoden kann man nicht an einfachen Beispielen im Unterricht aufzeigen.

Komplexe Systeme aber sprengen den Rahmen des Unterrichts.

Komplexe Systeme im Informatikunterricht
Beispiel OO-Modellierung



Komplexe Systeme im Informatikunterricht
Beispiel Entwurf von Datenbanken

table=file *attribute=field* *attribute value*

Catalogue

order number	name	other names	purity %	boiling point °C	...
43,919-3	methanol	methyl alcohol	99.93	64.7	...
27,074-1	ethanol	ethyl alcohol	90.0	78.0	...
29,328-8	1-propanol	propyl alcohol	99.5	97.0	...

relation

record=tuple

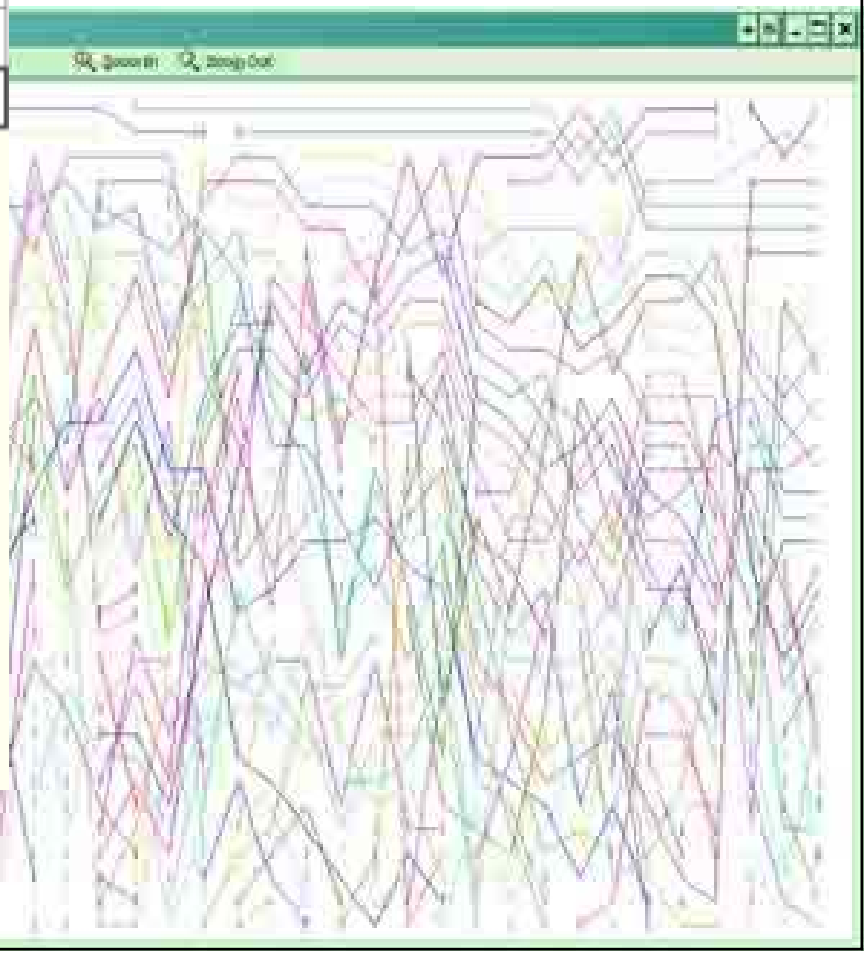
shopping cart

order number	customer	date	payment	status
27,074-1	406	31-01-2003	cash	paid
29,328-8	97	08-04-2003	credit	open

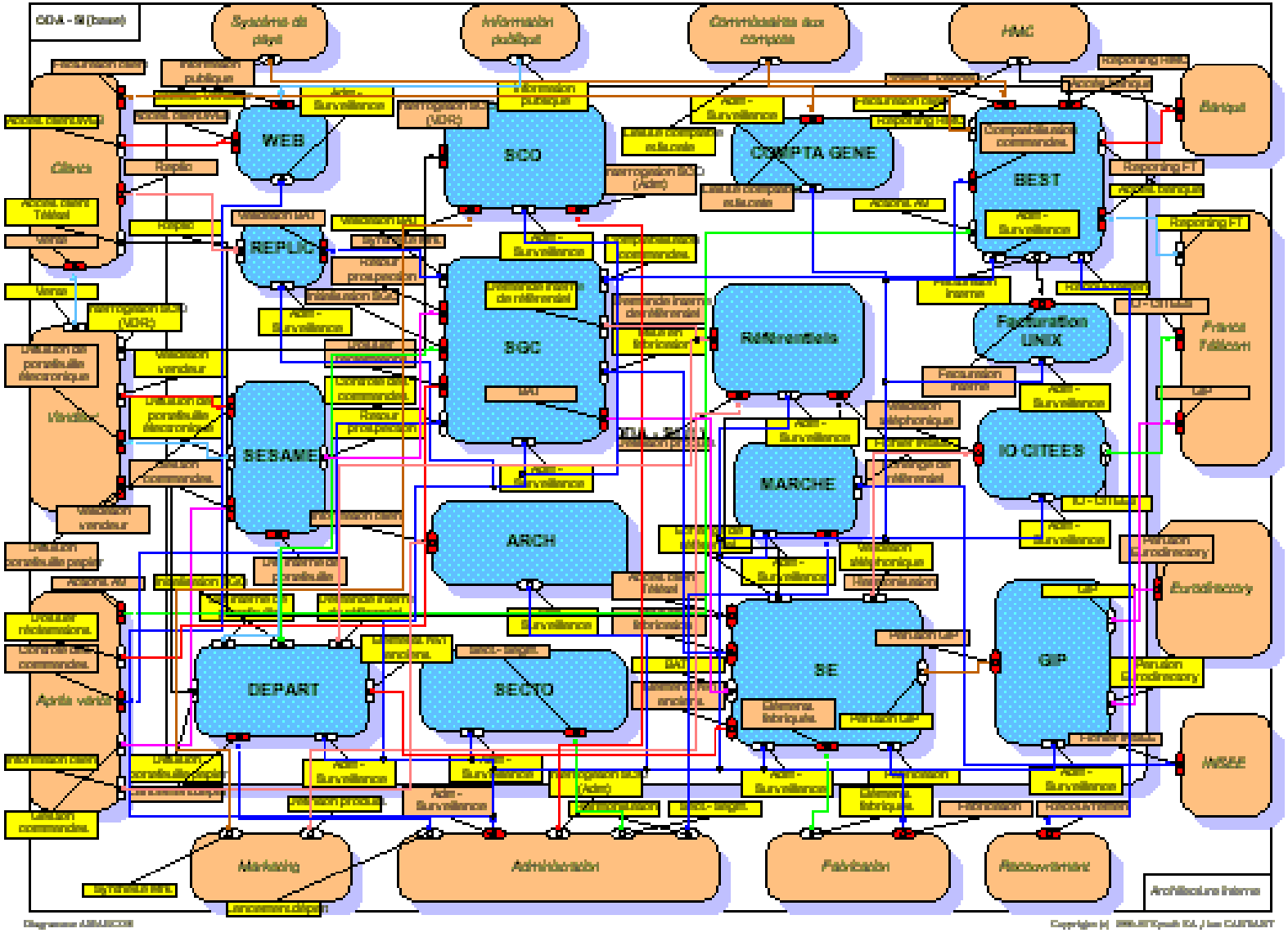
relation

customer list

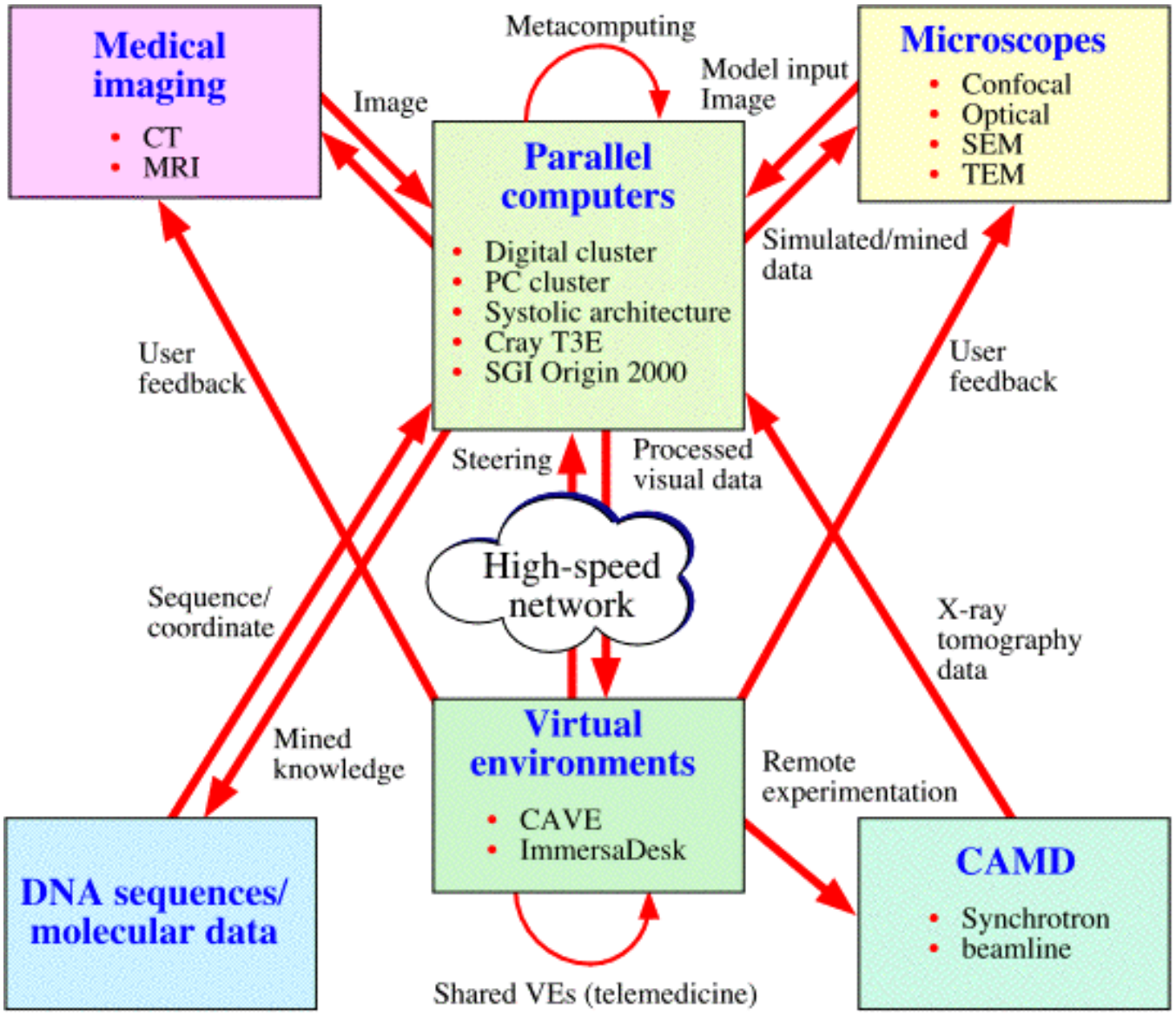
customer	name/company	street	...
97	Miller	Oxford Street	...
406	Jones	Sunset Blvd.	...
...	...	...	...



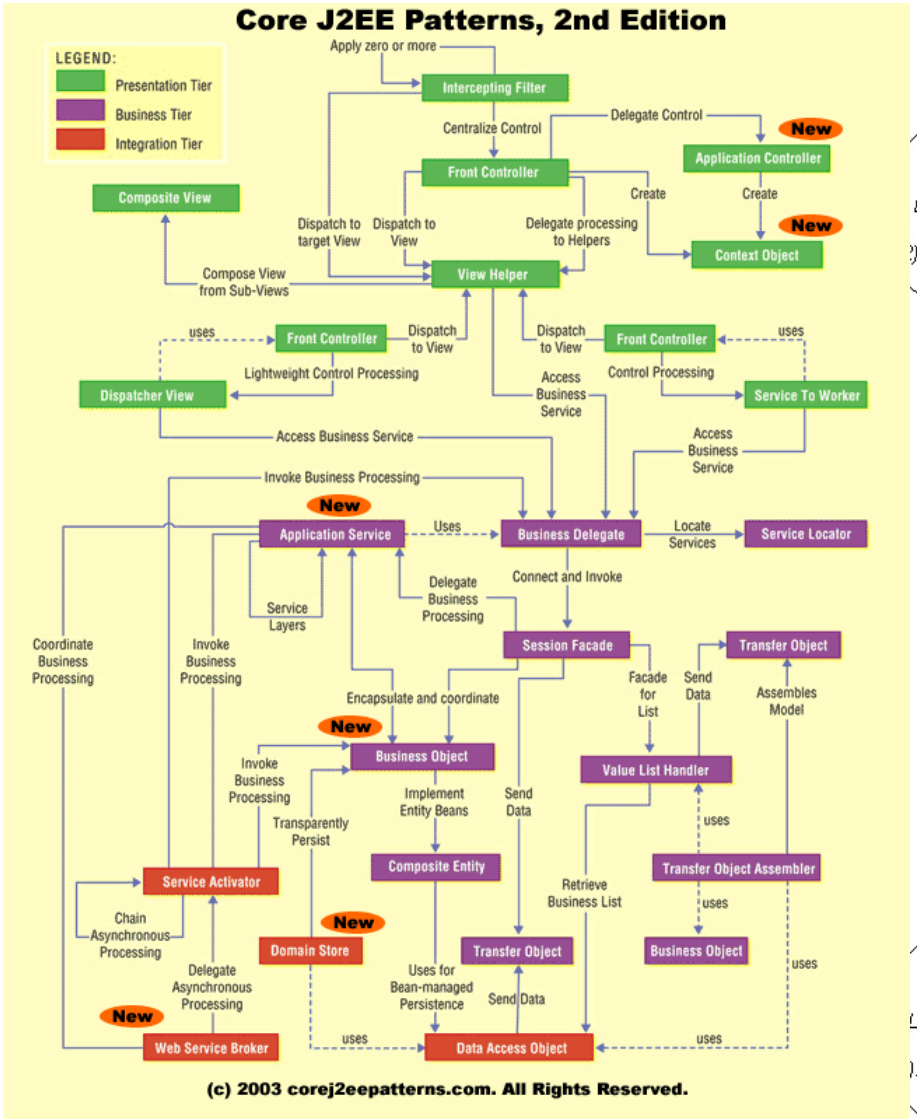
Komplexe Systeme im Informatikunterricht
Beispiel Informationsarchitektur

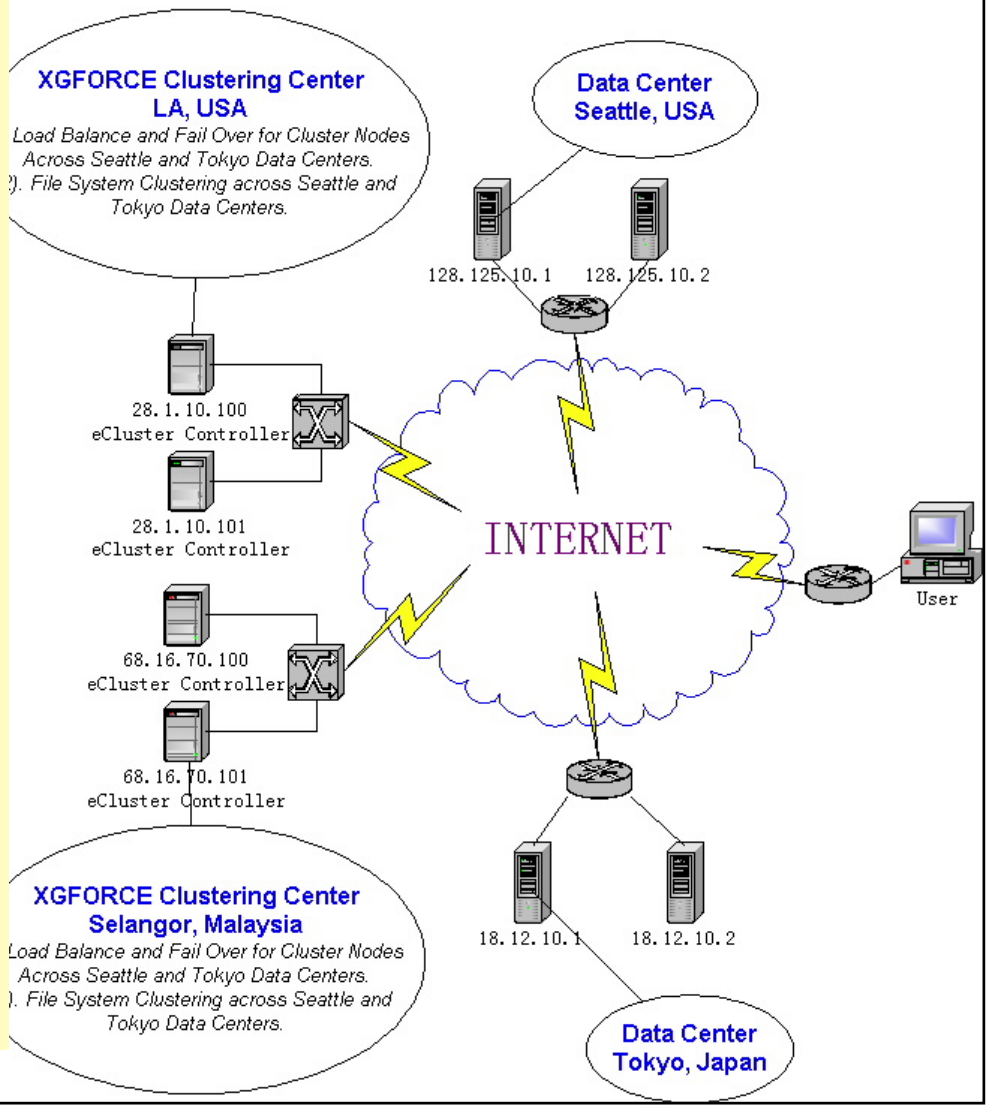


Komplexe Systeme im Informatikunterricht
Beispiel Concurrent Computing



Komplexe Systeme im Informatikunterricht
Beispiel J2EE, Load Balancing, ...





Komplexe Systeme im Informatikunterricht
Die Lösung des Problems

... ist uns nicht bekannt.

... wird immer wieder in schrillen Tönen angekündigt (Case Tools, dann Software Dekonstruktion, dann UML in der Grundstufe).

... gibt es vielleicht gar nicht?

... am ehesten noch Case Studies (Fallstudien)?

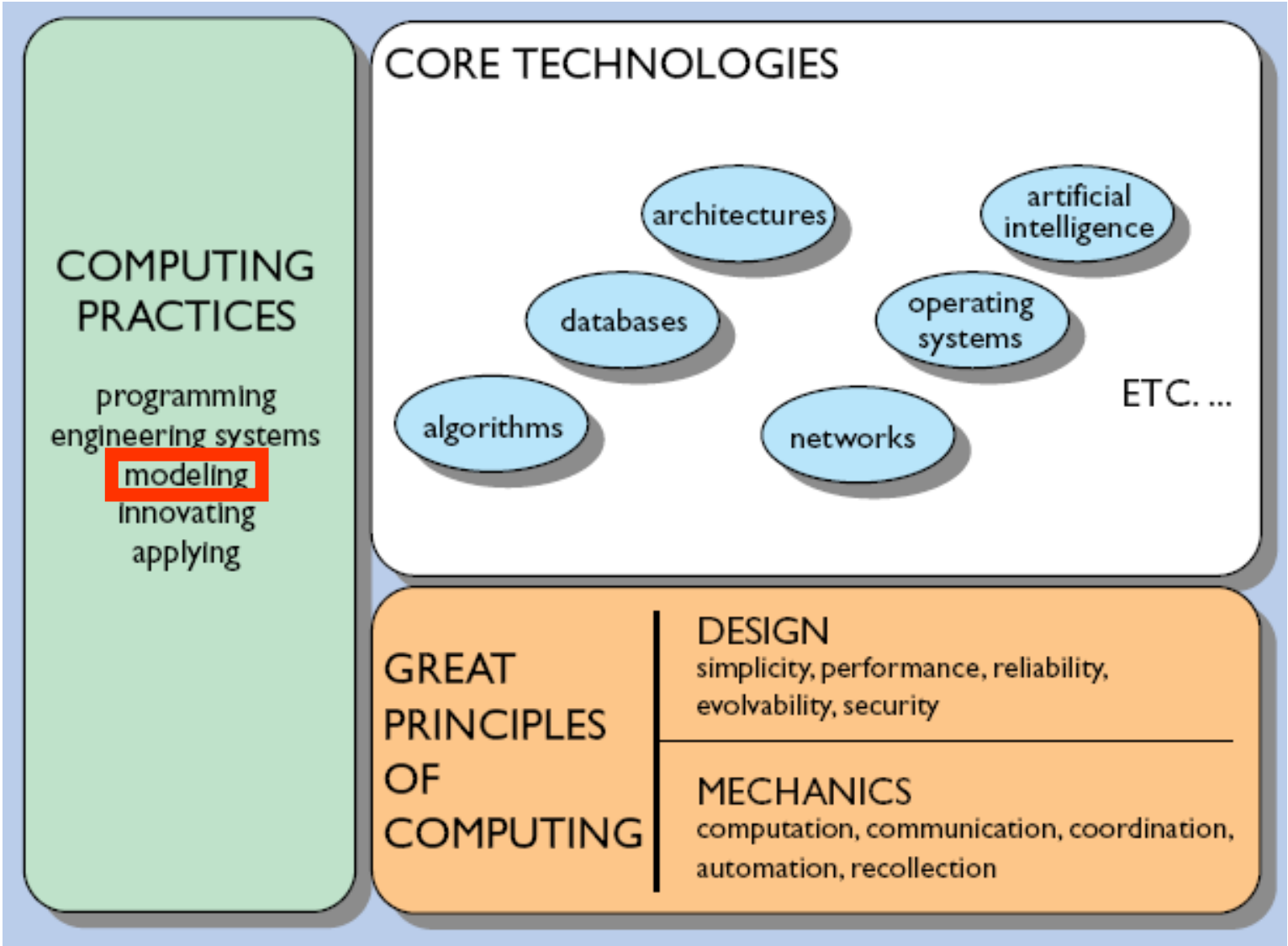
... oder inverted curriculum?

Fokus auf OOA/M/D/P... Wird im Informatikunterricht der Abstraktion ein zu hohes Gewicht beigemessen?

Abstraction is an important but not the defining principle of computing.

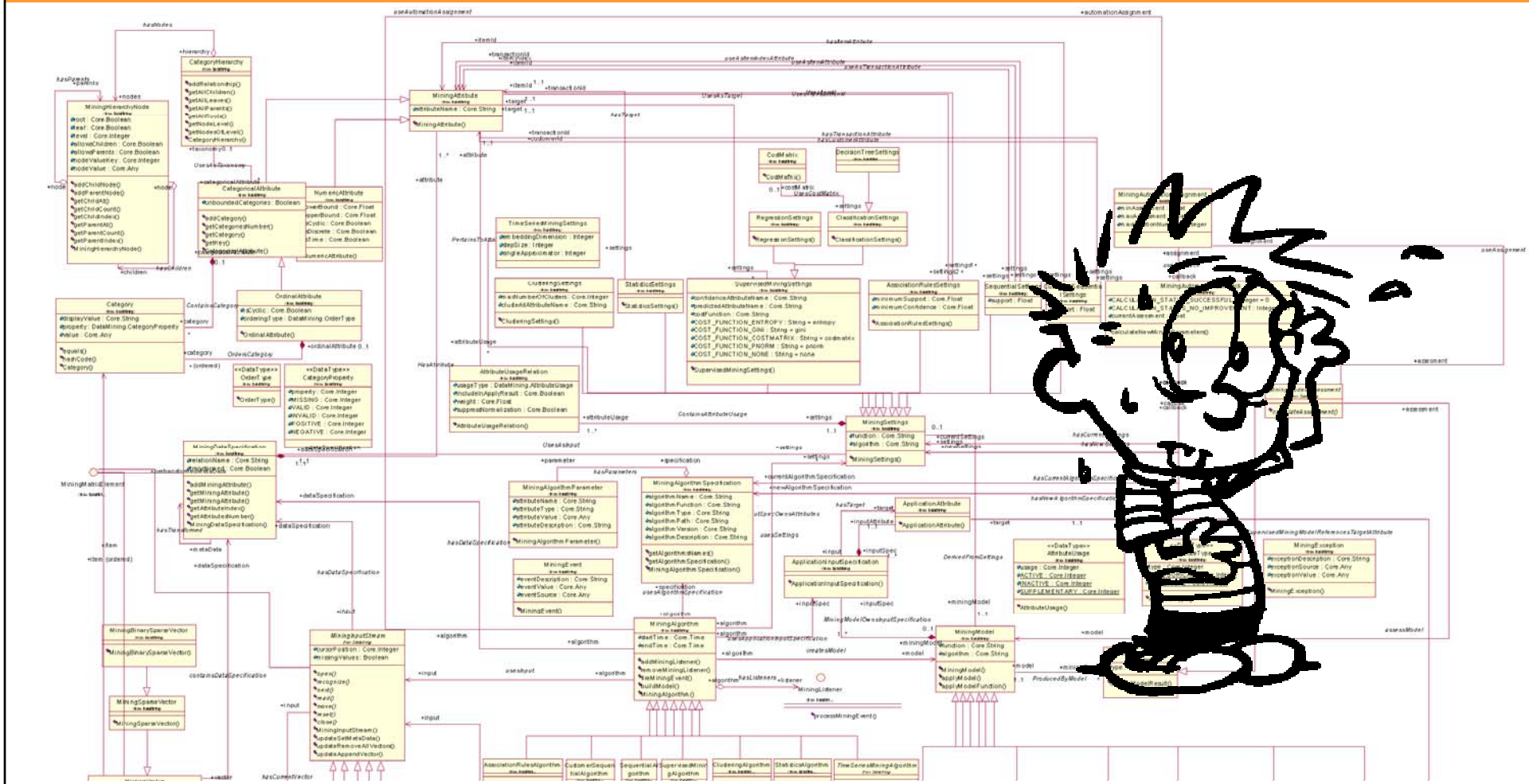
Peter Denning, Communications of the ACM, 2005; Look Beyond the Abstraction to Define Computing

„It is time to stop hiding the enormous depth and breadth of our field.“



Peter Denning, Communications of the ACM, 2005; Look Beyond the Abstraction to Define Computing

UML-Fever – auch an Schulen?

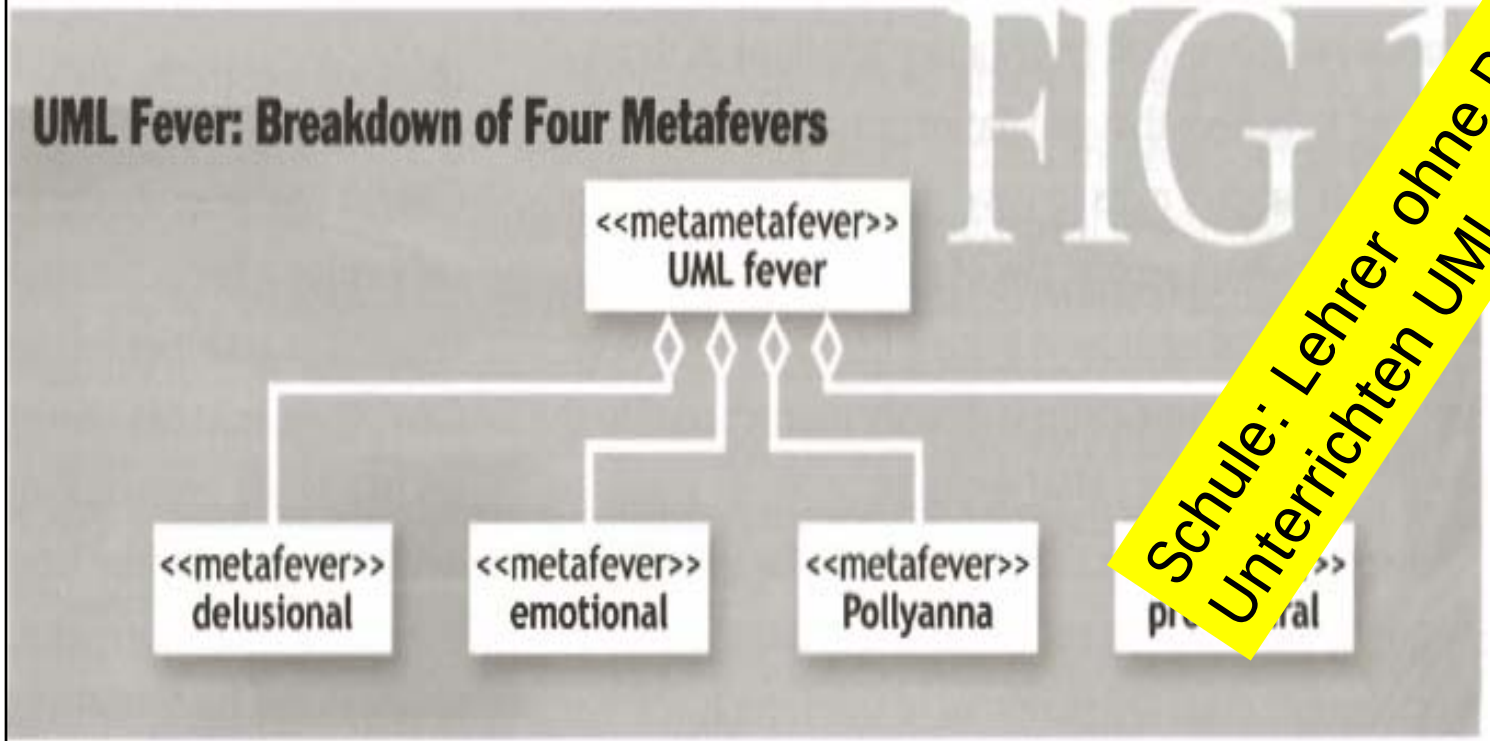


Alex E. Bell. Death by UML Fever. *ACM Queue - Tomorrow's Computing Today*, Vol.2, No.1, March 2004, p. 72-80.

UML-Fever: 4 Metafevers, 18 Fevers...

Grundlegendes Problem <<instanceof>> **Metaprobl**

„At the root of most UML fevers is a lack of practical experience in those individuals responsible for [...] soft development efforts.“

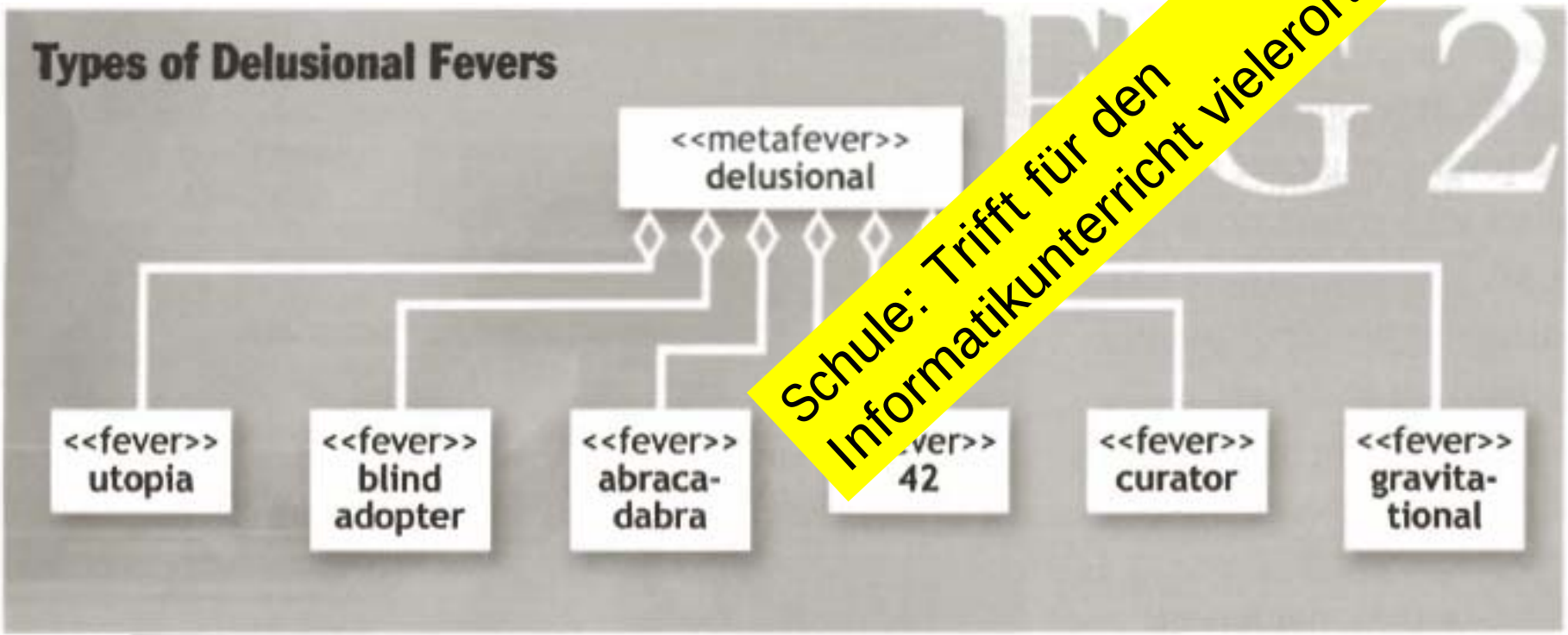


Schule: Lehrer ohne Projekterfahrung Unterrichten UML und OOP ...

Delusional: Den Sinn und „Wahn“sinn von UML richtig einschätzen...

Curator Fever <<instanceof>> **Delusional Metafever**

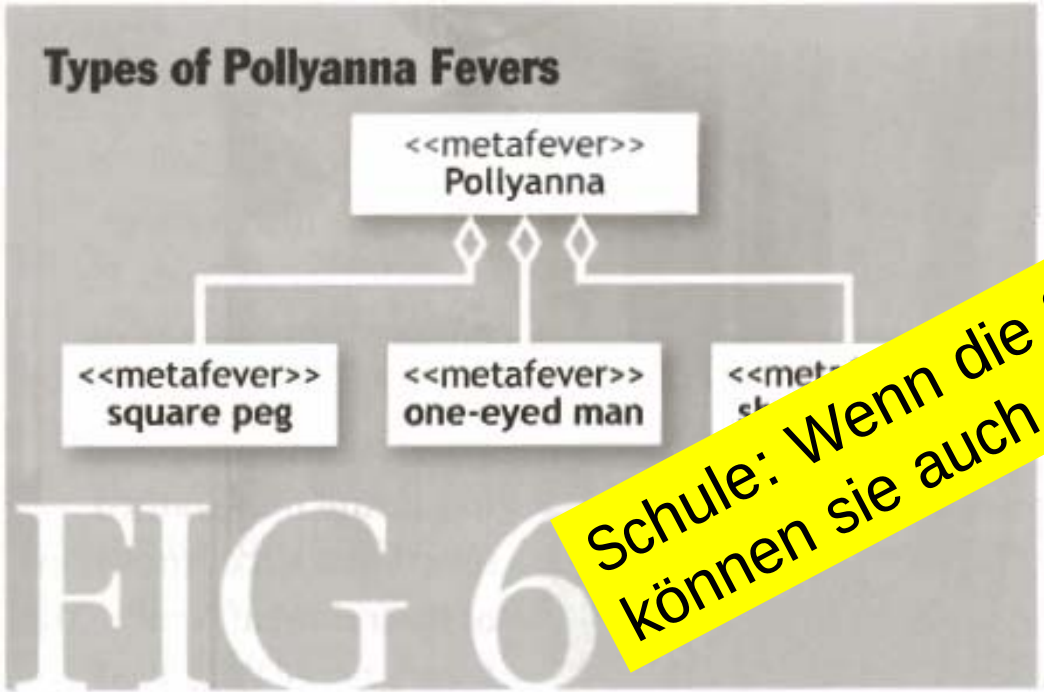
Victims believe „that production of UML diagrams, as opposed to the engineering content behind them, is the single most important activity in the software-development life

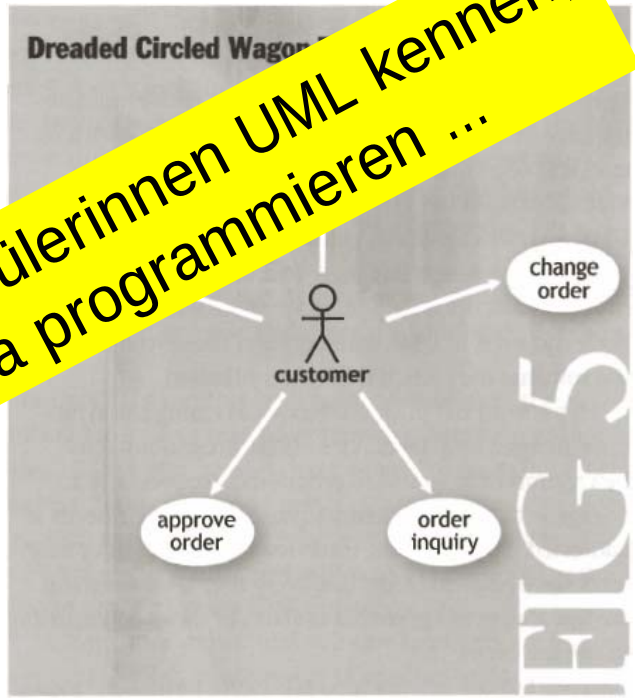


Schule: Trifft für den Informatikunterricht vielerorts zu ...

Pollyanna: Blinde Technik-Gläubigkeit

Shape Shifter Fever <<instanceof>> **Pollyanna Metafever**
Der Glaube, eine Einführung in UML und Design-Tools mache aus jemandem einen Software-Entwickler: „UML makes the designer“.





Schule: Wenn die Schülerinnen UML kennen, können sie auch Java programmieren ...

Beispiel aus „Teilapplikationen objektorientiert analysieren und implementieren“ (I-CH Modul 226)

Unsere Kundin, die Leiterin einer Jugendherberge, steht vor einem immer wiederkehrenden Problem.
Wenn eine Reisegruppe ankommt, dauert es immer sehr lange, bis die Mitglieder auf die Zimmer verteilt sind.

Die Jugendherberge besteht aus:
3 Zimmern mit 4 Betten,
2 Zimmern mit 5 Betten,
2 Zimmern mit 3 Betten und
1 Zimmer mit 2 Betten.

Die Leiterin ist der Meinung, dass dieses Problem mit einer IT-Teilapplikation gelöst werden kann. Sie gibt Ihnen den Auftrag, eine Teilapplikation objektorientiert zu erstellen und zu dokumentieren. Dazu gehören das Use Case Diagramm, das Klassendiagramm und das Sequenzdiagramm, das den Ablauf des Zuordnens der Mitglieder einer Reisegruppe zu den einzelnen Zimmern darstellt.

Beispiel aus „Teilapplikationen objektorientiert analysiebren und implementieren“ (I-CH Modul 226)

1. Notation und Symbolik von UML ist für die verlangten Diagramme angewendet.
2. Die Aufgabenstellung ist objektorientiert programmiert.
3. Die Diagramme sind überprüft und der Code der Teilapplikation ist methodisch getestet.
4. Analyse und Code sind unter Beachtung von Wartbarkeit und Nachvollziehbarkeit dokumentiert.

Diagnose: 42 Fever <<instanceof>> **Delusional Metafever**
Der Glaube, UML sei die Antwort auf Alles (aber zumindest sicher auf alle Software-Entwicklungs-Probleme).

Beispiel aus „Teilapplikationen objektorientiert analysieren und implementieren“ (I-CH Modul 226)

Verteilung von n **Gästen auf** x_i **Zimmer mit** y_i **Betten**
(wobei z_i Betten belegt sind und Männlein / Weiblein wohl getrennt werden müssten)

Ein rein kombinatorisches Problem:
Prozedural lösen. Oder einfach mit einer Excel-Tabelle.

Wenn schon Jugendherberge modellieren:
Klassischer Fall für Relationale Datenbank.

PS Wann objektorientiert?
Wenn mindestens einer der drei Eckpfeiler der Objektorientierung unabdingbar ist:
Abstraktion / Kapselung, Vererbung und Polymorphismus.

Zum Schluss die schlechte Nachricht: „The Fever is Real“

„The entertaining tenor of „Death by UML“ should not be inferrred to suggest that UML fever is an imaginery ailment. It is genuinely real, it is thriving, and its presence is causing cost and schedule trauma on many software programs right now.“

… und was ist mit den Lehrplänen… ?

Grady Booch. The Fever is Real. *ACM Queue - Tomorrow's Computing Today*, Vol.2, No.1, March 2004, p. 81.

Programmier-Einsteiger: Wie naheliegend sind OO-Modelle?

„[...] claims regarding the **“naturalness, ease of use, and power”** of the OO approach. [...]

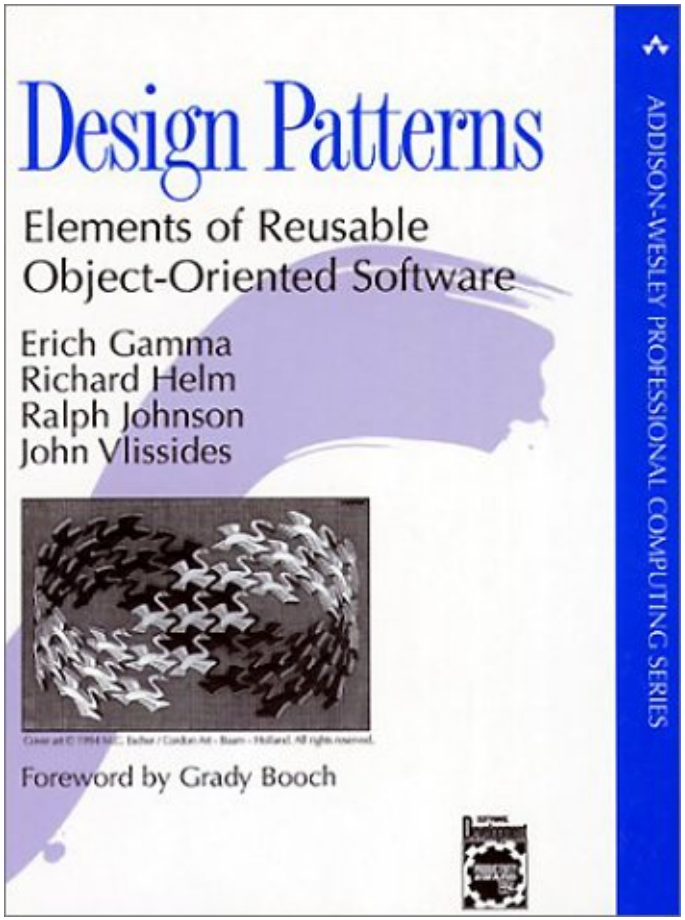
The papers reviewed do not support this position. They show that **identifying objects is not an easy process,**

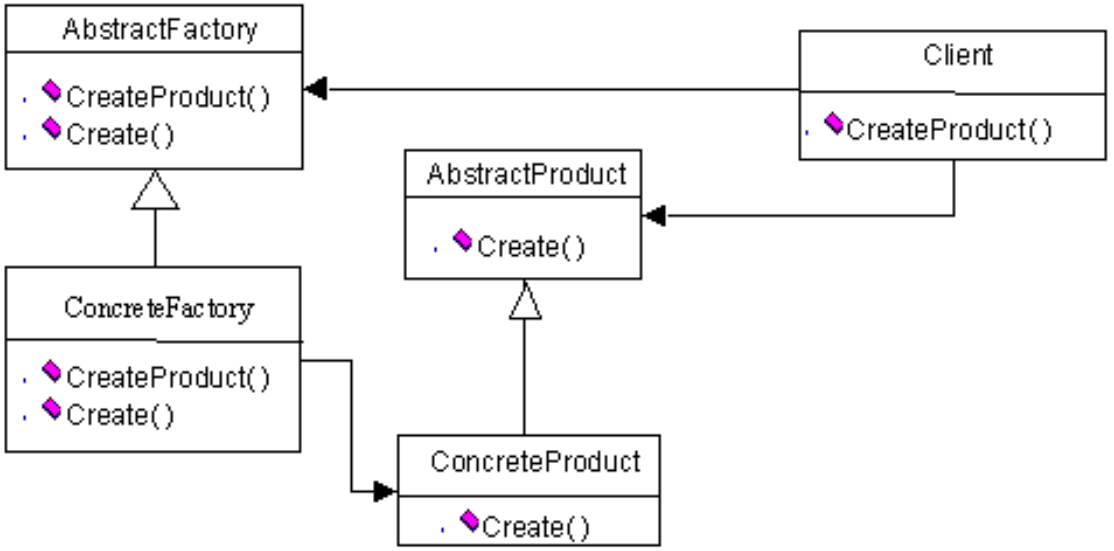
that **objects identified in the problem domain are not necessarily useful in the program domain,**

that the mapping between domains is not straightforward, and that novices need to construct a model of the procedural aspects of a solution in order to properly design objects/classes.

Anthony Robins, Janet Rountree, and Nathan Rountree. Learning and Teaching Programming: A Review and Discussion, *Computer Science Education,* 2003, Vol. 13, No. 2, pp. 137–172.

Programmier-Unterricht: Realistische OO-Beispiele sind komplex





Ziel: „Biete eine Schnittstelle zum Erzeugen von Familien verwandter oder voneinander abhängiger Objekte, ohne ihre konkreten Klassen zu benennen.“

Programmier-Unterricht: Geeignete OO-Beispiele für Einsteiger?

Beispiel: Kaffeemaschine, 2. Semester Inf.-Studium

„We want to design a **coffee machine**. A user can select different coffee products at different prices. The list of products is fairly extensive, such as decalf, espresso, and mocha. The machine maintains ingredients for each of the products. The user's choice is "made to order" from the ingredients. The products change from time to time, as do the prices, so it must be relatively easy to change these.

We want to design the interior working of the machine in an object-oriented way. Actually, all we need to design is a **simulation** of the machine. The simulation will then be used to control the physical devices. We do need an object "surrogate" for each physical device, however.

What objects are needed? How do they interact?

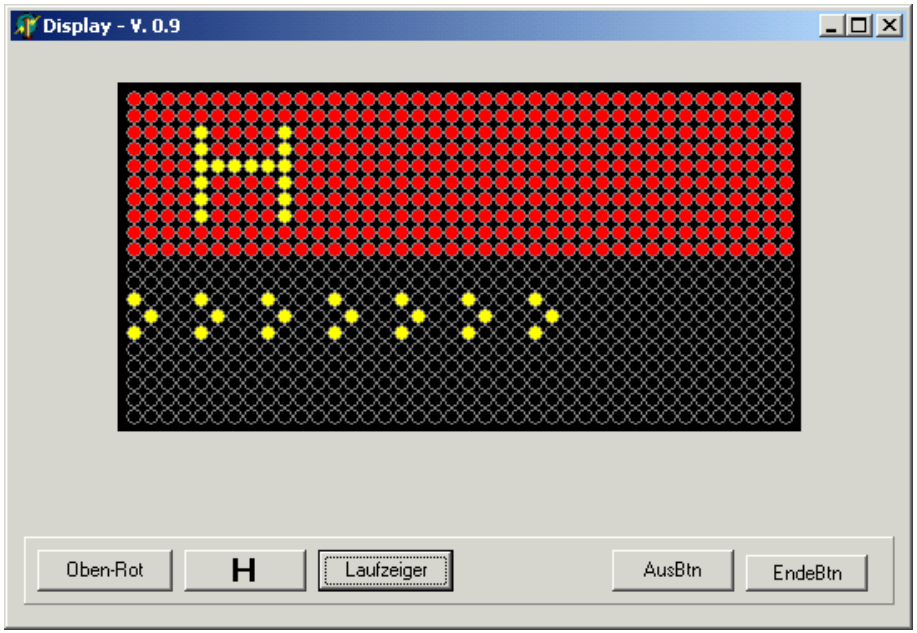
What messages (information flows) need to be implemented?

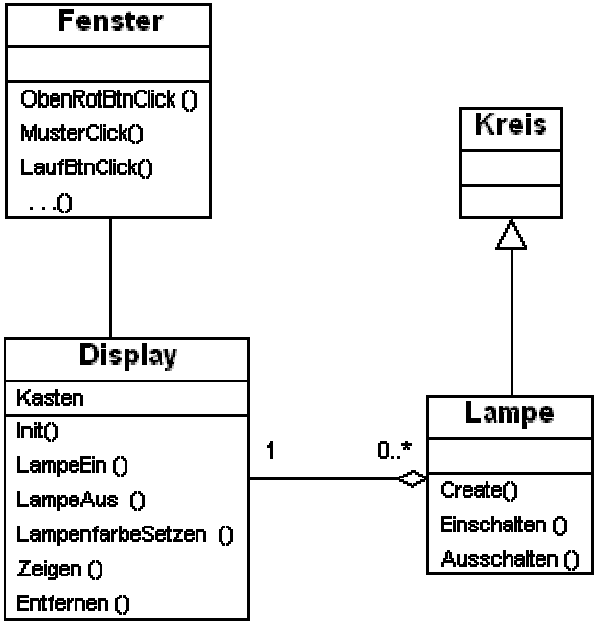
What classes are needed?“

Joseph Bergin, csis.pace.edu/~bergin/

Programmier-Unterricht: Geeignete OO-Beispiele für Einsteiger?

Beispiel: Punkte-Display, Gymnasium





http://www.oszhdl.be.schule.de/gymnasium/faecher/informatik/modellierung/uml.htm

Programmier-Unterricht: Wann Objekte? Zwei Ansätze

Bottom-Up: Von prozedural zu objektorientiert

- Mit dem „Programmieren im Kleinen“ anfangen
- Schwergewicht auf kleinen algorithmischen Beispielen
- Verstehen von Programmstrukturen etc.
- Objektorientierung erst später

Beispiel: Karas & Turtles, www.swisseduc.ch/informatik

Outside-In: Direkt objektorientiert einsteigen

- Von Anfang an Objekte **verwenden**
- Schwergewicht auf kleinen algorithmischen Beispielen
- Verstehen von Programmstrukturen etc.

Beispiel: Bertrand Meyer,
www2.inf.ethz.ch/~meyer/publications/teaching/teaching-psi.pdf

Programmieren im Grossen im Informatikunterricht

Gefahr beim Einstieg über's Programmieren im Kleinen:

- Studenten wählen einen Syntax-orientierten Ansatz beim Entwurf von Programmen
- Mangelndes Bewusstsein für grosse, reale Software-Systeme (kein Erkennen oder Wiederverwenden von komplexen Strukturen)

Wie "programmieren" Experten?

- Experten denken in grösseren konzeptuellen Strukturen, nicht in Syntax
- Experten entwerfen "Pläne"
- Experten denken vielseitig!
- Realität erfahrener Programmierer: top-down, bottom-up, middle-out, backtracking, revision , ...

Programmieren im Grossen im Informatikunterricht

OOP ist eine Konstruktionsmethode, um ausgehend von wiederverwendbaren Komponenten grosse Systeme zu entwerfen und implementieren.

Aufgabe eines Informatikingenieurs!

Richtiges OOP bedingt OO-Analyse und OO-Design. Allein die Verwendung von OO-Sprachkonstrukten ist keine objektorientierte Programmierung.

Programmieren im Grossen eignet sich nicht für eine Einführung in der Schule.

Man lernt nicht auf einem Airbus fliegen!

Objektorientierter Ansatz bringt trotzdem im Unterricht Vorteile. Objekte, Klassen und Vererbung begegnen uns in vielen Anwenderpaketen.

Programmieren im Grossen im Informatikunterricht

Einstieg über Analyse und Bearbeitung von dokumentierten Systemen

Anhand von Fallstudien

- Designentscheidungen erkennen
- verschiedene Lösungsansätze gegeneinander abwägen
- gewählte Lösungen dokumentieren und verteidigen
- erkennen, dass auch Experten unterschiedliche Lösungen favorisieren

Durchführung kleinerer Software-Projekte im Team

Programmieren im Grossen: Programme erweitern

Today generally students are required to develop programs from scratch and are rarely asked to use and modify existing complex programs that could not be redeveloped ab initio.

Jacques Cohen, Updating Computer Science Education, Communications of the ACM, June 2005

Programmieren im Grossen: Programme erweitern

Riesige Datenmengen stehen in neuen Formen zur Verfügung (z.B. Daten von Suchmaschinen oder Satelliten)

Neue und komplexe Programme, welche nicht von Grund auf neu entwickelt werden können (z.B. Google, DNA-Analyse Programme)

Jacques Cohen, Updating Computer Science Education, Communications of the ACM, June 2005

www. swisseduc.ch / informatik / karatojava

www. swisseduc.ch / informatik / turtles

Probieren geht über Studieren!

Raimond Reichert - raimond.reichert@acm.org